Titre: Title:	Modèle de données pour la représentation d'informations de simulation dans un processus d'optimisation de composantes hydrauliques
Auteur: Author:	Horea Adrian Iepan
Date:	2004
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Iepan, H. A. (2004). Modèle de données pour la représentation d'informations de simulation dans un processus d'optimisation de composantes hydrauliques [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7398/

NOTE TO USERS

UNIVERSITÉ DE MONTRÉAL

MODÈLE DE DONNÉES POUR LA REPRÉSENTATION D'INFORMATIONS DE SIMULATION DANS UN PROCESSUS D'OPTIMISATION DE COMPOSANTES HYDRAULIQUES

HOREA ADRIAN IEPAN
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2004

Your file Votre référence
ISBN: 0-494-01342-7
Our file Notre référence
ISBN: 0-494-01342-7

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

MODÈLE DE DONNÉES POUR LA REPRÉSENTATION D'INFORMATIONS DE SIMULATION DANS UN PROCESSUS D'OPTIMISATION DE COMPOSANTES HYDRAULIQUES

présenté par: IEPAN Horea Adrian

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. TRÉPANIER Jean-Yves, Ph.D., président

M. GUIBAULT François, Ph.D., membre et directeur de recherche

M. GAGNON Michel, Ph.D., membre

REMERCIEMENTS

Je tiens à exprimer ma profonde gratitude à mon directeur de recherche, le professeur François Guibault, pour sa disponibilité et ses conseils avisés. Je souhaite également le remercier pour m'avoir laissé une grande liberté dans mon travail tout en me faisant bénéficier de ses connaisances dans le domaine de l'optimisation et integration multidisciplinaire.

Je tiens à remercier particulièrement à Robert Magnan, cercheur à IREQ (l'Institut de recherche d'Hydro-Québec), qui à toujours eu la patience de repondre à mes questions. Ce travail n'aurait pu aboutir sans son précieuse aide dont l'expérience m'a beaucoup apporté.

Je tiens également à remercier tous les membres du projet OPALE et, tout particulièrement, Marie Gabrielle Vallet, Jean Francois Dube et Charles Brassard qui ont toujours su être disponibles pour répondre à mes nombreuses questions.

Enfin, je remercie à Sebastien Laflamme, Paul Labbé et Julien Dompierre qui même s'ils ne faisait pas partie du projet OPALE ont su être disponibles pour répondre à mes questions.

RÉSUMÉ

Le travail présenté dans ce mémoire vise à définir l'infrastructure nécessaire pour faciliter l'échange de données entre différents outils d'analyse et d'optimisation multidisciplinaire. L'automatisation du processus d'optimisation du design des aubes de turbines hydrauliques est un processus complexe, qui intègre plusieurs sous-processus issus de différents domaines. L'intégration de plusieurs sous-processus implique l'échange de données entre ceux-ci.
Un modèle de données CFD pour la représentation de l'information numérique dans le domaine spécifique des turbines hydrauliques a été développé. Le modèle de données a été développé en partant du concept abstrait de définir et de structurer les données CFD définies par la norme CGNS.

Enfin, pour valider le modèle de données développé, dans le contexte de l'optimisation du design de turbines hydrauliques, nous avons montré à travers une application réelle une façon de calculer le rendement d'une turbine en utilisant un fichier XML pour conduire le processus.

En étudiant les approches employées par les standards de représentation de données numériques dans les domaine de la CFD et la CAO, nous avons proposé une approche flexible qui permet l'imposition de contraintes de structure et de contenu sur les données, tout en respectant la norme CGNS.
Ensuite, on a développé une méthode basée sur XML pour gérer la façon d'imposer des contraintes sur une base de données CFD.
Partant de celle-ci, on a montrés comment on peut utiliser la nouvelle approche pour amener des structures de données générées par des différents outils commerciaux ou *« maison »* dans une forme standard définie a priori.

Les résultats démontrent l'efficacité de l'approche proposée à imposer des contraintes

sur des bases de données dédiées à la CFD ou la CAO.

ABSTRACT

The work presented in this paper aims to define the infrastructure necessary to facilitate the data exchange between various tools of analysis and multidisciplinary optimization. The automation of the optimization process of the hydraulic turbine blades design is a complex process, which integrates several under-processes resulting from various fields. Hence, the integration of several under-processes implies data exchange between those under-processes.
A CFD data model for representation of numerical information in the specific field of the water turbines was developed. The data model was developed on the basis of the abstract concept to define and structure CFD data defined by the CGNS standard.

To validate the developed data model in the context of the water turbines design optimization it was showed, through a real application, a way to calculate the output of a turbine by using a XML file to lead the process.

Studying the approaches employed by the standards in the field of CFD and CAD data representation, it is proposed a flexible approach which allows imposing constraints at the level of the structure and contents, while respecting the CGNS standard. Furthermore, a XML based method to manage the way of imposing constraints on a CFD data base, was developed. It was showed also, how this new approach can be used to bring generated by various commercial or *in house* tools data structures in the definite standard form.

The results show the effectiveness of the approach suggested imposing constraints on CFD or CAD dedicated data bases.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ANNEXES

CHAPITRE 1

INTRODUCTION

L'évolution récente des processus de développement d'un produit se caractérise par une complexité toujours grandissante des méthodologies de conception et la nécessité de s'accommoder de cycles de design de plus en plus courts. Les pratiques en ingénierie subissent ainsi des changements profonds à tous les niveaux de production et dans tous les champs de l'industrie.

Bien que ces changements se fassent sentir dans tous les domaines de l'industrie et, entre autres, dans le secteur de l'automobile, leur impact est beaucoup plus évident dans les industries où les produits sont fabriqués en séries limitées, comme c'est le cas de l'industrie aéronautique ou dans la conception des grandes turbines hydrauliques. Les changements dans la pratique de l'ingénierie traditionnelle ont initié une forte poussée au niveau de la recherche et du développement (R&D) dans le domaine de l'automatisation et de l'optimisation multidisciplinaire des designs. Cette transition - qui en même temps nous conduit vers des approches de design beaucoup plus compétitives - se fait déjà sentir, en amenant des changements profonds aux niveaux organisationnel et technique dans les entreprises.

Une approche de design traditionnelle est formée d'un ensemble de petits groupes de design, qui sont chacun spécialisés dans une discipline spécifique : l'hydraulique, les structures, les vibrations, la fabrication, etc. Traditionnellement, le travail sur un projet est organisé d'une manière séquentielle, chaque discipline étant considérée comme un processus en soi. Les disciplines sont ordonnées d'une manière ayant comme but la minimisation des retours arrière. Chaque discipline incorpore un niveau de connaissance sous la forme de modèles simplifiés des disciplines subséquentes. Les restrictions et les plages des modifications autorisées au niveau du

design sont imposées par les premières disciplines. Ce modèle en cascade donne la possibilité de produire des designs dans un temps très court, en minimisant l'interaction entre les concepteurs. L'applicabilité d'un tel modèle est cependant restreinte à un contexte où les disciplines impliquées dans la conception d'une composante peuvent être ordonnées d'une manière séquentielle. À certaines étapes de design, on peut se retrouver avec des modèles qui ne respectent pas les conditions imposées, ce qui peut entraîner la nécessité de re-designer complètement un ou plusieurs composants. Une telle approche n'offre pas la possibilité d'explorer systématiquement plusieurs solutions de design et l'intégration de nouvelles disciplines est très difficile. Une autre approche consiste à considérer les activités de design dans un cycle itératif. Sachant que toutes les disciplines qui font partie d'un processus d'optimisation du design peuvent s'influencer mutuellement, les ingénieurs ont amorcé le développement de nouvelles approches capables d'administrer des contraintes spécifiques issues de diverses disciplines.

Pour de meilleurs résultats, une telle approche doit être intégrée dans un système de design automatisé et il doit disposer d'un système d'échange de données efficace, afin de permettre aux ingénieurs de partager l'information vite et sans pertes.

En plus des transformations subies par le processus de design en ingénierie, de nouvelles disciplines comme la simulation numérique à haute fidélité émergent et elles doivent être intégrées dans le processus de design. Les industries commencent à utiliser de plus en plus d'outils de simulation et d'analyse numérique pour le prototypage virtuel, par rapport aux tests faits en laboratoire. La performance des outils développés et l'exactitude des modèles numériques donnent des résultats qui sont comparables aux résultats expérimentaux, ce qui permet de valider et d'imposer ces nouvelles disciplines.

Les analyses en dynamique des fluides numérique (computational fluid dynamics - CFD) et en dynamique des structures numérique (computational structural dynamics - CSD) font maintenant partie du quotidien des ingénieurs. À cause de la

complexité des projets, le nombre d'ingénieurs impliqués dans ces activités d'analyse et de design augmente.

Dans ce contexte, la nécessité de disposer d'une infrastructure pour faciliter l'échange et le partage des données devient prépondérante.

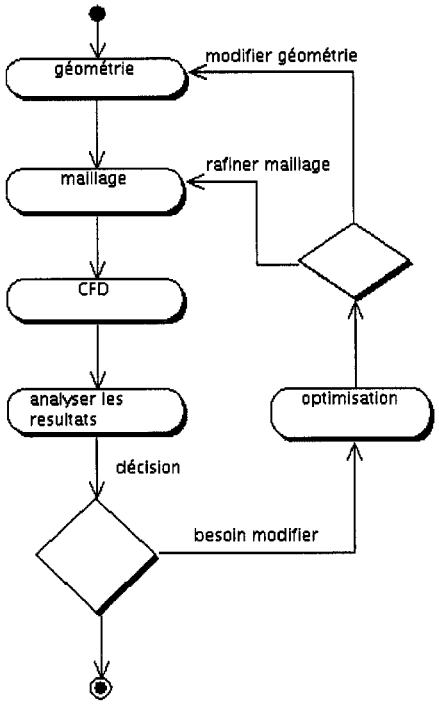


FIGURE 1.1 Le flots de données dans un processus d'analyse numérique

Dans le domaine de la CFD, les processus d'analyse numérique peuvent être représentés d'une manière relativement uniforme. Les entités qui composent un tel modèle de processus d'analyse numérique incluent : la construction du modèle géométrique, la génération du maillage, la résolution numérique des équations et l'analyse (et la comparaison) des solutions (fig. 1.1).

Avant d'aborder les différents aspects présents dans un processus d'analyse en ingénierie, la modélisation et la simulation numérique, il convient de préciser la terminologie, les définitions de base et quelques notions utilisées de façon générale dans ce type de processus.

La modélisation géométrique représente la description d'un espace 2D ou 3D à l'aide

de primitives géométriques de bas niveau, comme des sommets, arêtes et surfaces. Par maillage, on entend la discrétisation d'un espace paramétrique (2D ou 3D), ayant comme résultat un réseau de recouvrement pour le domaine donné, domaine formé par des éléments qui respectent certaines propriétés :

- Deux éléments adjacents ne peuvent avoir qu'une arête (face) en commun.
- Si une arête (ou une face) d'un élément n'appartient pas à un autre élément, elle appartient à la frontière.

Il faut tenir compte de plusieurs points, afin de savoir si les modèles de représentation de toutes ces données rendront réellement possibles les différents traitements envisagés par la suite. Pour qu'un tel système de stockage puisse remplir correctement son rôle, il doit :

- permettre de garder des informations géométriques et topologiques,
- pouvoir définir correctement les noeuds du maillage,
- prendre en compte les conditions limites,
- pouvoir définir et retracer correctement les solutions numériques
- pouvoir visualiser les résultats numériques,
- permettre d'autres traitements liés à la nature du problème.

Les entités composant un tel système d'analyse sont en général des logiciels développés par différents concepteurs ou des logiciels «maison» développés pour des besoins spécifiques. Les concepteurs essayent de donner à ces logiciels un certain degré de standardisation, en les dotant de la capacité d'utiliser plusieurs types de formats (comme entrée et sortie). À cause des différences introduites par chaque concepteur au niveau de l'implémentation, la compatibilité n'est souvent pas satisfaite. Plusieurs approches sont utilisées aujourd'hui dans le but de contourner le problème de compatibilité au niveau des échanges et du partage des données entre applications. L'approche la plus employée pour contourner ce problème est basée sur des traducteurs (fig. 1.2). La sortie de chaque processus passe par un traducteur qui, en

traduisant le format d'entrée dans le format voulu, assure le lien avec le processus suivant. Un processus de traduction, dans ce contexte, est plutôt un processus

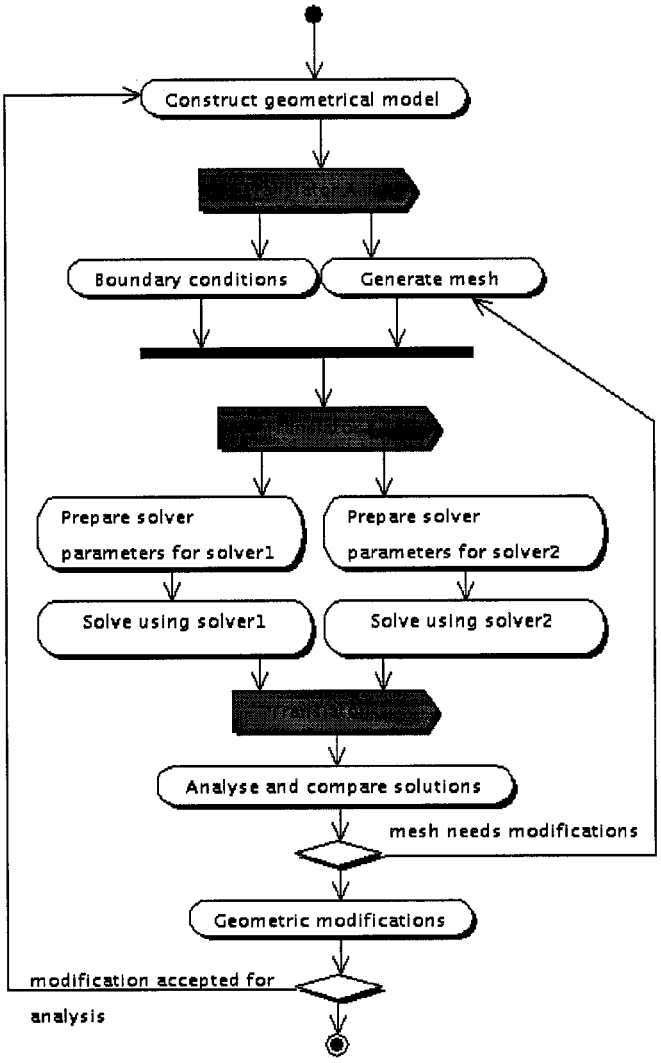


FIGURE 1.2 Le processus d'analyse basé sur des traducteurs

d'adaptation d'un format à un autre qu'une traduction. C'est une adaptation, parce qu'une traduction implique le transfert d'information d'un système de représentation dans un autre sans perte d'information, ce qui implique que même si les modes de représentation des données sont différents, les données doivent être les mêmes (ce qui n'est pas le cas de tous les format de représentation utilisés aujourd'hui). Une

variante de cette approche proposée et implémentée est basée sur des traducteurs répartis sur le WEB. Les applications de traduction sont réparties sur Internet, et les utilisateurs les utilisent au travers d'une interface Java. Le site « aspire » le fichier en le traduisant. Une fois l'opération de traduction terminée, le nouveau fichier dans le format voulu sera prêt à être téléchargé à partir de ce site.

L'approche basée sur des traducteurs présente plusieurs désavantages : la génération de données redondantes (les mêmes données en plusieurs formats), l'utilisation inutile de ressources CPU, le temps de développement gaspillé (n formats nécessitent n *x (n-1)* traducteurs), la possibilité de pertes de données dues à la traduction et la perte de fiabilité des formats. Le problème est loin d'être résolu, et la solution basée sur des traducteurs est de plus en plus remise en question.

Une alternative qui gagne constamment en popularité consiste à utiliser un format de représentation assez général pour pouvoir garder toutes les informations nécessaires au processus d'analyse numérique.

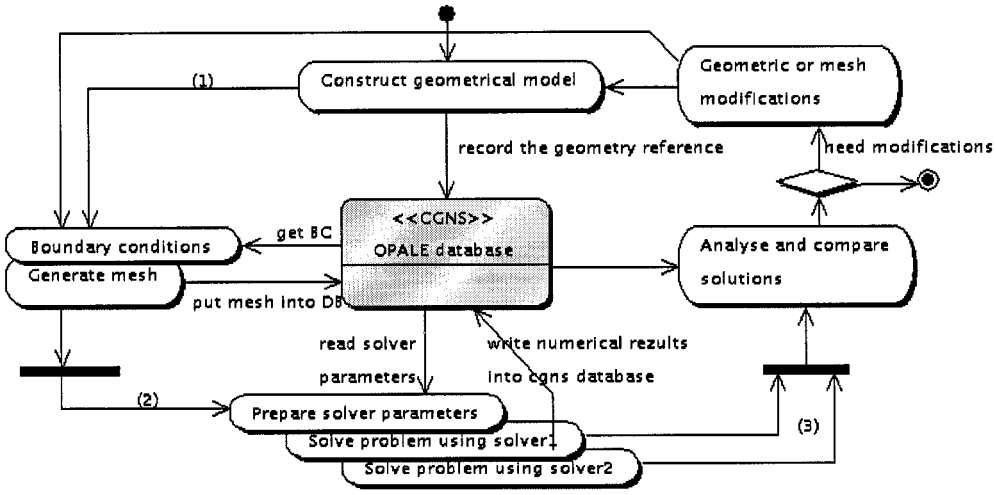


FIGURE 1.3 Le processus d'analyse basé sur un format de représentation général

Un tel modèle, (fig. 1.3) qui englobe tant le modèle de données CFD que le modèle CAO, pourra être facilement utilisable dans un processus d'optimisation automatisé. Des données redondantes, et les traducteurs qui les génèrent vont disparaître. Les

processus pourront échanger des données facilement.
Un tel format est appelé Méta-Modèle CAO ou Méta-Modèle CFD. De plus, le Meta-Modèle developpé devrait permettre d'assurer l'échange des données CFD entre des applications et des outils d'analyse et de design, commerciaux ou *«maison»*. Ceci fournira un cadre suffisant pour considérer les activités d'analyse et de design dans un cycle itératif d'optimisation.

Le but de ce projet est d'évaluer les approches existantes du point de vue de l'analyse numérique et de l'optimisation multidisciplinaire et de proposer un mode d'utilisation et un modèle qui combine ces approches pour créer un Méta-Modèle incluant à la fois la CAO et les données CFD.
Le présent projet étant développé en collaboration avec l'IREQ (l'Institut de recherche d'Hydro-Québec), le modèle de données CFD développé a comme but de donner l'infrastructure pour représenter des informations numériques dans un domaine bien particulier, celui des turbines hydrauliques. Il est évident que le modèle bâti aura des caractéristiques particulières issues de ce domaine.
L'idée derrière ce projet est de développer une méthode d'optimisation des aubes de turbines hydrauliques. L'optimisation de la géométrie des pales doit se faire en cherchant à maximiser le rendement des turbines hydrauliques. Pour que le processus d'optimisation soit efficace, ses composantes doivent être intégrées dans une boucle itérative et automatisée. Les composantes sont les applications et les outils d'analyse commerciaux et «maison».
Même si théoriquement on peut considérer les composantes dans une boucle automatisée, le processus d'analyse ne peut pas encore être totalement automatisé parce que certaines des tâches liés à l'analyse doivent être accomplies par des ingénieurs. Nous cherchons à automatiser le plus possible les phases d'analyse, parce que l'intervention humaine dans un tel processus constitue non seulement un facteur de ralentissement du processus, mais également une importante source d'erreurs.

Dans ce contexte, la nécessité de disposer d'une infrastructure pour faciliter l'échange et le partage des données devient prépondérante.
Deux étapes devait être accomplies a fin d'établir cette infrastructure :

- définir un modèle de représentation de données qui fournira le cadre nécessaire pour intégrer les étapes de design dans une boucle automatisée
- trouver une façon efficace d'exprimer le modèle

Le développement d'un modèle de représentation de données numériques CFD, dans le contexte particulier des turbines hydrauliques, permettant d'assurer l'échange et le partage des données basé sur la norme *CGNS*, constitue le principal objectif de ce mémoire. Parce que la morme CGNS definit un format qui ne garde pas les géométries mais des références vers des entités géométriques, les possibilités de garder les liens avec le modèle CAO seront également étudiées.
Le modèle de données CFD est développé en se basant sur la norme CGNS, et en utilisant le langage XML.
Du point de vue de l'implantation physique, le modèle doit respecter les requis suivants :

- Fournir un mécanisme efficace pour vérifier et contrôler la façon dont le résultat d'une simulation CFD est structuré dans la base de données.
- Permettre de vérifier que la base de données CFD est complètement définie avant de l'utiliser dans un processus d'optimisation ou d'analyse numérique.
- Permettre de facilement modifier la structure de la base de données en fonctions des besoins spécifiques des applications d'analyse.
- Prévoir un mécanisme permettant d'accéder facilement aux entités géométriques sur lesquelles le maillage est défini.
- Utiliser un format libre de droit.

Bien que presque tous les requis puissent être rencontrés grâce à la norme STEP, nous avons préféré XML pour plusieurs raisons : la difficulté d'interpréter les fichiers de données STEP, les voix de plus en plus nombreuses qui essayent de montrer com-

ment passer de STEP à un autre format de description de données (en particulier XML), le fait que le format STEP n'est pas libre de droit, etc.
Le langage XML dispose d'un mécanisme qui lui permet facilement de rencontrer les exigences du projet. Tous les problèmes liés à la vérification et à la validation de la structure et du contenu du fichier sont résolus à l'aide du dictionnaire de données défini pour ce type de structure de données.
Le modèle de données CFD-XML est formalisé par un ensemble de règles de grammaire, le dictionnaire du modèle. La grammaire d'une base de données CFD doit être rigoureusement respectée.
L'utilisation de l'un ou l'autre de ce formats pour stocker effectivement la base de données représente une matérialisation du modèle. Donc, ce qui doit être vérifié du point de vue de la conformité est une instance de ce modèle, une matérialisation du modèle. En exprimant la structure d'une instance en XML, le problème de la validation est réglé par le dictionnaire.

Parmi les formats étudiés, on peut énumérer : IGES, STEP, CGNS, PIE, XML et VRML. L'évaluation de ces formats sera traitée dans le chapitre 2, en essayant de cerner leurs fonctionnalités et leur complémentarité potentielle.
En ce qui concerne la simulation, le stockage des données avant la simulation, le stockage des résultats de simulation et les liens avec la géométrie, les possibilités offertes par CGNS et XML seront présentées (les types de maillage qui peuvent être traités, les conditions-frontière, les types de liens pouvant être conservés, la taille des fichiers et la possibilité ou la nécessité d'utiliser un ou plusieurs fichiers).
Dans le troisième chapitre, un modèle pour l'échange des données numériques (CFD), dans le contexte spécifique des turbines hydrauliques, basé sur CGNS est développé.
Dans le chapitre 4, le concept et la réalisation pratique d'un processus de mise en forme standard d'une base de données CGNS, basée sur XML en utilisant des gram-

maires (DTD) spécialement développées pour des données CFD est aussi présenté. En ce qui concerne la validation du modèle développé, dans le chapitre 5, la mise en oeuvre d'une application de calcul des pertes basée sur le format CGNS est présentée. Étant donné les facilités offertes par XML, le processus de recherche des données dans la base de données CGNS est conduit par XML.

CHAPITRE 2

REVUE DES STANDARDS POUR LA REPRÉSENTATION DES DONNÉES D'ANALYSE NUMÉRIQUE (CAO - CFD)

Dans les années 60 et 70, chaque système CAO utilisait son propre format de fichiers pour représenter les informations de design d'un produit. Le transfert d'informations se faisait en produisant des copies de design sur papier, et par la suite en introduisant les informations afférentes à la main dans le système de la discipline suivante. Ce mode de fonctionnement était une source considérable d'erreurs et produisait des données redondantes, en plusieurs formats. Comme le nombre de systèmes CAO augmentait, le besoin d'échanger des informations entre plusieurs systèmes s'est mis à augmenter. Progressivement, le transfert des informations entre les systèmes s'est fait en utilisant des outils de traduction maison. Même si les outils de traduction pouvaient être rapides et assez précis, cette approche était inefficace à cause du grand nombre de traducteurs nécessaires : N x *(N-1)* , où N représente le nombre de systèmes impliqués.

Avec le nombre de plus en plus significatif d'applications en ingénierie qui utilisent la CAO, la nécessité d'avoir un standard pour représenter les données CAO est devenu impérative. Dans ce contexte, vers la fin des années 70, le premier standard pour la représentation des informations CAO, IGES est né. Depuis, de nouveaux standards ont émergé, tels STEP, XML, VRML, etc.

Sous la poussée de l'industrie, le nombre et la variété des outils de simulation numérique s'accroissent constamment. En ce qui concerne la gestion des informations issues des outils de simulation, plusieurs standards émergent, chacun avec ses points forts et ses points faibles.

Parmi les formats les plus utilisés aujourd'hui dans le domaine de l'analyse numérique, on peut nommer CGNS [7] et plus récemment XML [9].

En essayant de cerner leurs fonctionnalités, ces formats seront décortiqués dans ce qui va suivre. Les aspects qu'on essayera de mettre en évidence relativement à chaque standard sont les suivants :

- le champ d'application du standard
- la structure des données
- les éléments de syntaxe
- les limitations

Plus particulièrement, la capacité du format STEP à représenter des données géométriques ainsi que les possibilités offertes par le format CGNS pour la représentation de données de simulation numérique seront étudiées.

Du point de vue conceptuel, les standards étudiés dans ce projet peuvent être classifiés en deux grands groupes : des standards pour la représentation des géométries et des standards pour la représentation des solutions numériques.

2.1 IGES - Initial Graphics Exchange Specification (CAO)

La première version de IGES a été introduite en janvier 1980 et elle est devenue une norme ANSI en septembre 1981 [12]. IGES a été l'une des premières normes pour l'échange de données entre différents systèmes et visait initialement l'échange de données de dessins techniques. Les versions subséquentes de la norme ont étendu les types de données supportées (courbes, surfaces, élément fini, solides, représentation frontalière).

IGES utilise des entités pour représenter et communiquer les données afférentes

d'un produit. L'entité est l'unité fondamentale dans un fichier IGES et elle possède assez d'informations pour supporter les besoins de presque tous les systèmes CAO d'aujourd'hui. Les entités peuvent être classifiées en entités géométriques et entités non-géométriques. Les entités géométriques représentent les définitions du corps physique : points, courbes, surfaces et solides. Les entités non-géométriques donnent des informations concernant le point de vue, le dessin, les dimensions, les notations, etc.

Chaque fichier IGES comporte 5 sections obligatoires :

-Début (START)
-Globale (GLOBAL)
-Répertoire (DIRECTORY ENTRY)
-Données (PARAMETER DATA)
-Fin (TERMINATE)

Chaque ligne représente une entrée composée de 80 caractères.

La section *Début* fournit un prologue non codé du fichier IGES. Elle doit contenir au moins une ligne d'enregistrement.

La section *Global* contient une description du logiciel qui a créé le fichier et d'autres informations utiles pour la lecture, parmi lesquelles on peut énumérer : le système d'unités, la précision, les dimensions maximales du dessin, etc.

La section *Répertoire* (Directory Entry) donne une description générale de chaque entité du fichier. Elle contient aussi des informations non géométriques.

La section *Données* du fichier contient les paramètres associés à chaque entité. Ces paramètres peuvent être des informations géométriques de même que des pointeurs. Les paramètres sont placés en format libre jusqu'à la colonne 64. La colonne 65 est toujours occupée par un espace blanc. Les colonnes de 66 à 72 contiennent le numéro

de séquence de la première ligne de l'entité dans la section Répertoire. La lettre P pour Parameter Data est inscrite dans la colonne 73. Les colonnes 74 à 80 doivent contenir le numéro de séquence de la ligne.

Dans la section *Fin* il n'y a qu'une seule ligne. Cette section se situe à la dernière ligne du fichier et elle est divisée en dix champs de huit colonnes. La lettre T pour Terminate est inscrite dans la colonne 73.

En ce qui concerne les entités supportées par le format IGES, on y trouve des entités :

- *géométriques* (arc de cercle, plan, ligne, point, spline paramétrique de courbe, spline paramétrique de surface, B-Spline rationnelle de courbe/surface, etc.)
- *d'annotation* (dimension angulaire, dimension de diamètre, flèche, surface sectionnée, etc.)
- *de structure* (définition d'association de style de ligne, syntaxique, définition du maillage d'une sous figure, vue, etc.)

Les problèmes de ce format sont conceptuels. Le fait qu'il n'y ait pas un modèle de données formel induit des ambiguïtés dans certains cas. En faisant abstraction de la syntaxe complexe d'un fichier IGES - qui requière par exemple 80 caractères sur chaque ligne - et de l'organisation très complexe des données - ce qui rend les fichiers très lourds à lire, interpréter et corriger -, les concepteurs de systèmes CAO ont « personnalisé » le standard par rapport à chacun de leurs systèmes, ce qui fait que l'échange de données est devenu très difficile. Le résultat est que l'échange des données géométriques entre deux systèmes différents - qui en fait utilisent tout les deux le standard IGES pour représenter les données géométriques, n'est pas assuré, encourant souvent une perte d'information et des coûts de manipulation des fichiers. Un exemple concret d'un fichier IGES se trouve à l'annexe II.

2.2 STEP - ISO 10303 Standard for the Exchange of Product Model Data

La norme STEP [10] a été développée au cours des vingt dernières années pour répondre à des besoins spécifiques de normalisation des données de définition de produits. Ainsi, cette norme se veut un moyen de communication afin de décrire et de permettre l'échange de données relatives aux produits tout au long de leur cycle de vie, indépendamment des différents systèmes de production utilisés.

La norme englobe toutes les données relatives au produit, en commencent par les caractéristiques géométriques jusqu'à la nomenclature et en passant par les méthodes et les outils de fabrication et les protocoles d'applications pour son implantation.

Le standard est défini par une collection de parties, chacune publiée séparément, qui tombent dans l'une des catégories suivantes : méthodes de description, ressources intégrées, protocoles d'application, méthodes d'implantation ou méthodes de test.

La norme est basée sur une architecture en trois couches : le modèle de référence, qui peut définir un nombre de modèles typiques spécifiques à des applications individuelles, la couche logique, qui définit le langage formel EXPRESS, et la couche physique qui définit la structure d'un fichier de communication, nommé fichier STEP.

En étudiant la philosophie de STEP, on y trouve deux composantes de base à l'aide desquelles la norme est bâtie : il s'agit d'EXPRESS, qui est un langage formel pour la description de données, et les protocoles d'application (Application Protocols - AP) qui entourent le langage.

2.2.1 EXPRESS

EXPRESS - spécifié dans la norme ISO 10303-11 [2] - est un langage formel expressif et mature pour la description des données. Il est structuré à l'aide de schémas qui représentent le modèle du produit. Un schéma est constitué d'entités qui sont des objets de base, et qui encapsulent des attributs et des contraintes.

L'exemple suivant montre un point décrit en langage EXPRESS :

```
SCHEMA exemple geometrie;
TYPE
    lengthMeasure = NUMBER;
END TYPE;
ENTITY point
    SUPERTYPE OF ( ONEOF (cartesianPoint) )
END ENTITY;
ENTITY cartesianPoint
SUBTYPE OF (point);
    X_coordinate : lengthMeasure;
    Y_coordinate : lengthMeasure;
    Z_coordinate : OPTIONAL lengthMeasure;
END ENTITY;
END SCHEMA;
```

2.2.2 Les protocoles d'application (Application Protocols - AP)

Chaque AP est un document formel qui décrit les activités présentes dans le cycle de vie d'un produit et c'est la seule façon d'implémenter STEP. Par exemple, *STEP*

AP part 21[10] définit le format du fichier d'échange. N'importe quelle application doit être en mesure de lire et écrire ce type de fichier d'échange. La partie qui décrit la géométrie est STEP AP part 42 (ISO 10303-42) [10]. Les entités topologiques avec lesquelles le standard STEP travaille sont : le sommet, le sommet-point qui est un sous-type du sommet, le noeud, le noeud-courbe, le noeud-orienté, la boucle, la face, la sous-face, la face orientée, des faces connectés, etc.

2.2.3 Limitations de STEP

Deux limites importantes de STEP sont : d'une part pour pouvoir comprendre et utiliser STEP, il faut avoir de bonnes connaissances du modèle EXPRESS et ensuite il utilise trop les ID numériques d'objets qui ne sont pas persistants. De plus, les fichiers de données sont des fichiers sans aucune hiérarchie, contenant seulement une grande liste d'objets.

2.3 Passage de IGES à STEP

Plusieurs compagnies, comme STEP Tools[3], ProSTEP[2], EuroSTEP [5], ont développés des outils capables de faire la traduction du format IGES au format STEP.

STEP Tools Inc[3] a développé un outil capable de traduire la géométrie IGES en format STEP. Toutes les géométries sont converties en NURBS (Non-Uniform Rational B-Splines) en laissant les autres ressources géométriques offertes par STEP non utilisées. Il y a des traducteurs créés pour des applications particulières qui traduisent des fichiers IGES en fichiers STEP. Basu et Kumar [14] ont créé un traducteur pour l'analyse par éléments finis qui extrait les entités de maillage - noeuds et éléments - d'un fichier IGES, et les convertit en entités STEP pour la mèthode des éléments finis.

Pour faciliter l'importation et l'exportation d'entités IGES vers les produits CAO, des compagnies comme AutoCAD[2], PRO/Engineer [2] et CATIA[2], ont développé des traducteurs *"maison"* pour faciliter la conversion d'IGES dans leurs propres formats. Ces outils ne peuvent pas être utilisés indépendamment pour l'échange des données. Il sont toujours liés à leurs pré-processeurs qui écrivent leurs fichiers CAO en fichiers IGES ou STEP.

Bhandarkar [16] et al., en se basant sur l'AP 202 [2] de STEP, ont développé un outil de traduction d'IGES à STEP. Ils se sont concentrés sur la géométrie, la topologie et le dessin technique. Le traducteur analyse le fichier IGES, en le traduisant en un fichier STEP (AP202) correspondant. En utilisant le langage EXPRESS, ils ont créé des schémas qui décrivent les spécifications d'IGES. Ce schéma contient une entité nommé directory-entry qui est un super-type abstrait de geometric-entity, constructive-solide-geometry-entity, boundary-representation-entity, annotation-entity et structure-entity. Par exemple, les entités IGES *rational b-splines* courbes et surfaces sont traduites en STEP en entités composés, formés par des *b-splines* (courbes et surfaces) avec des noeuds, et *rational b-spline* (courbes et surfaces). Ceci est rendu nécessaire parce que IGES impose les noeuds et les poids pour l'entité *rational b-splines*, tandis que STEP ne pose aucune restriction. Donc, pour faire la conversion complète il faut utiliser toutes les entités.

Le processus de traduction, laborieux, est décomposé en 6 étapes :

1. Le fichier IGES est lu dans une base de données ROSE.
2. Le traducteur analyse cette base de données, en traduisant chaque entité IGES conformément à l'AP202.
3. Les nouvelles entités AP202 sont maintenant ordonnées pour former la représentation du modèle en *geometrically-bounded-surface-shape et advanced-brep-shape-representation* en fonction de l'existence ou non d'informations topologiques dans le fichier.

4. Les annotations et la géométrie sont groupées en fonction de leurs types.
5. Les données AP202 sont sauvegardées dans la base de données ROSE.
6. Le format ROSE est converti dans le format STEP correspondent.

Par ailleurs, les entités CSG (Constructive Solid Geometriy) ne peuvent pas être converties en format STEP directement, étant donné que STEP utilise uniquement une représentation frontalière (Boundary Representation - BREP) pour représenter les solides. Il faut donc convertir les solides CSG en utilisant un modeleur solide comme ACIS pour interpréter l'arbre CSG et faire la transformation en B-Rep. Pour cette raison, plusieurs systèmes de traduction ne supportent pas les entités de type CGS. De plus, les entités éléments finis ne sont pas supportées par les traducteurs. Theresa L.Benyo[15] décrit une approche pour créer un environnement capable d'accéder aux données géométriques et de les restituer dans des formats propres aux applications de simulation numérique CFD directement à partir d'un fichier CAO. L'approche utilise deux composants : le *Project Integration Architecture-PIA* et la*Computational Analysis Programming Interface-CAPRI.* Le PIA est une architecture orientée objet pour capturer et intégrer les éléments liés aux activitées de recherche en aérospatial.

CAPRI est une interface de programmation créée en vue d'acquérir des géométries directement de fichiers CAO.

Mais CAPRI est en fait une collection de librairies spécifiques à chaque vendeur de systèmes CAO. Jusqu'à maintenant, CAPRI comprend des librairies qui supportent les formats de Unigraphics, ProEngineer, CATIA, FELISA, Computervision's CADDS et SDRC's I-DEAS. On a donc toujours le même problème : trop de formats pour représenter la même chose, et des formats qui ne sont pas compatibles.

2.4 XML eXtensible Markup Language

Une approche plus récente issue de l'informatique et de la manipulation d'information sur le web tend à s'imposer comme méthode universelle de représentation des données.

XML [9] est un métalangage de balises qui a été créé pour décrire des données. À l'époque où il a été créé, le but était de proposer un langage générique utilisable sur le Web, mais à court terme, à cause de sa généralité, il est devenu le point d'attraction entre autres pour la manipulation des données géométriques, où des standards comme IGES et STEP régnaient. Étant un métalangage, les balises qui le composent ne sont pas prédéfinies, donnant à l'utilisateur la possibilité de les définir en fonction de ses besoins à lui, ce qui lui confère sa grande généralité. Le langage XML dispose d'un mécanisme lui permettant de décrire des types de données en posant des restrictions aux niveaux structurel et syntaxique. Pour décrire des types de données, il utilise des dictionnaires des données (Document Type Definition-DTD) ou des schémas (XSD). La majorité des documents XML sont conçus à partir de DTD ou de XSD particuliers. Pour travailler avec des données CAO ou CFD, il est nécessaire de définir son propre dictionnaire de données à partir duquel les données seront structurées.
XML peut être défini comme une technologie qui peut créer la liaison entre des structures qui sont compréhensibles par des humains et celles qui sont interprétables par des machines.

Les points forts de XML, sont : la puissance de modélisation qui permet de décrire des données géométriques et CFD, la lisibilité, la facilité d'interprétation et la bonne intégration avec le Web.

2.4.1 Les analyseurs d'un fichier XML

Dans cette section, nous décrirons les approches conceptuelles afin d'analyser un fichier XML. Il existe deux méthodes d'analyse d'un fichier XML : DOM (Document Object Model) et SAX(Simple API for XML). De façon générale, aucune des deux méthodes ne se démarque de l'autre, chacune ayant des points forts et des faiblesses.

Le DOM définit un ensemble standard de commandes que l'analyseur XML doit intégrer afin que les données présentes dans un fichier XML soit accessible. Un analyseur XML qui utilise le DOM extrait les données d'un document XML en utilisant un ensemble d'objets qui sont structurés sous forme de noeuds d'un arbre. Chaque item d'information, qu'elle soit une balise, un attribut ou une donnée présente dans un fichier XML est traduite par l'analyseur DOM dans un noeud de l'arbre. L'approche nous donne une vue globale sur le document XML, avec la possibilité d'accéder à n'importe quelle donnée du document, mais il est très clair, que la méthode est gourmande en ressourcées, particulièrement au niveau de la mémoire.
Même si on appelle le SAX un analyseur des fichiers XML, il implémente en fait une interface de programmation d'applications (API) basée sur des évènements. Lorsque l'analyseur rencontre le début ou la fin d'une balise, il génère un événement capté par le gestionnaire d'événements actuellement enregistré. L'approche SAX demeure de bas niveau, et nécessite une plus grande responsabilité du point de vue de l'implémentation, puisque c'est l'implémentation qui doit gérer l'état de l'analyseur. Du point de vue de la performance, l'approche SAX dépasse largement celle du DOM, autant par rapport au temps de traitement nécessaire que de l'espace mémoire utilisée. Contrairement à l'approche DOM, le temps de traitement est linéaire, tandis que l'espace mémoire est constant.

2.4.2 La grammaire ou les dictionnaires des données

Avec un dictionnaire DTD[8] ou XSD[11], on peut décrire de manière stricte la structure d'un fichier XML. Les dictionnaires définissent un ensemble des conditions que les analyseurs doivent vérifier dans le but d'accepter ou non un document XML. Avec les outils fournis par les analyseurs, on pourra alors vérifier si un fichier XML est "valide" ou non selon une grammaire donnée. Il est très important d'avoir un bon dictionnaire et donc de consacrer du temps à sa conception. C'est lui qui nous permettra de construire des documents XML valides.

Les DTD sont les dictionnaires les plus utilisés dans le but de définir la structure d'un document XML. Il y a deux façons d'ajouter un DTD à un document XML. Il peut être défini soit à l'intérieur du document XML soit dans un fichier à part. Cette dernière solution est la plus fréquente car la plus pratique.

Un DTD a deux buts principaux :
- Fournir une syntaxe pour décrire les détails concernant la structure logique d'un document XML.
- Fournir une syntaxe pour définir la structure physique d'un document XML.

La définition de la structure logique d'un document XML, est implantée dans un DTD à l'aide d'entités de type *ELEMENT* et *ATTLIST*. On indique par exemple, quels types d'élements peuvent contenir un autre type d'élément. En ce qui concerne la structure physique d'un document XML, le seul mécanisme utilisé est celui basé sur des entités (*ENTITY*). Les entités donnent la possibilité de définir des dictionnaires (DTD) à partir d'autre dictionnaires.

Par exemple, dans le cadre de ce projet, nous avons défini un dictionnaire de données DTD qui décrit des données CFD (XCFDML), où nous ne spécifions pas quelle structure logique le document XML doit avoir. Les valeurs qu'un attribut *name* d'une entité spécifiée peut prendre sont définies comme des entités paramètres dans

un autre DTD, et les valeurs que les attributs *nodeData* peuvent prendre sont définies dans un autre DTD, ce qui facilite la compréhension et la maintenance. Une des faiblesses du DTD's est le fait qu'il ne supporte pas la définition de type de base ou de types définis par l'utilisateur.

Le XSD (schéma) représente un mécanisme flexible pour décrire des structures des données (comme le DTD) avec l'aptitude de déclarer des nouveau types de données, une facilité de modélisation qui permet de créer des structures de données ayant un comportement semblable.
Les avantages les plus significatifs d'un XSD par rapport à un DTD sont qu'un XSD :

- est défini en utilisant le langage XML lui même ;
- implémente le concept d'encapsulation ;
- implémente le concept d'héritage ;
- supporte des contraintes de type ;
- peut définir des types de données.

La philosophie du XSD est basée sur la notion de types et d'éléments. Les types peuvent être prédéfinies (booléen, binaire, flots, etc) ou définis par l'utilisateur et peuvent être utilisées pour définir des éléments. Il n'y a pas ici une relation un à un entre les types et les éléments comme dans le cas des DTD.

Même si les XSD sont plus puissantes en termes de description des données et des contraintes quelles peuvent spécifier, dans le cadre de ce projet nous avons utilisé un dictionnaire de type DTD. Cette décision est due essentiellement au fait que au moment où nous avons commencé le travail, un dictionnaire de type DTD était déjà disponible. Nous avons utilisé ce dictionnaire comme modèle, et nous avons développé le notre, qui est bien différent de l'original.
Un exemple du dictionnaire (DTD) est présenté à l'annexe II.

2.5 De STEP à XML

Pour montrer la puissance du standard XML pour décrire des entités géométriques de base avec lesquelles tous les systèmes CAO travaillent (point, arête, courbe, surface, volume, etc.), nous allons faire un parallèle entre la façon de représenter ces entités, dans un des systèmes de représentation de données les plus utilisés aujourd'hui (STEP), et XML, qui est un langage de descriptions de données.
Même si STEP est une norme bien définie en ce qui concerne la description des données CAO, qui dispose d'un langage de description de données (EXPRESS) et qui est la plus utilisée aujourd'hui pour décrire des géométries, il y a déjà plusieurs approches largement utilisées pour faire le passage de STEP à XML.

Pour passer de STEP à XML, il y a deux approches qui sont couramment utilisées : *Late Binding* et *Early Binding* [?]. Si, dans le cas du Late Binding, les noms qui composent le vocabulaire XML correspondent directement aux types de données et attributs définis dans le modèle EXPRESS, dans le cas de Early Binding, on utilise plutôt un format spécifique pour chaque modèle, ce qui nous conduit à des DTD beaucoup plus typés et beaucoup plus grands, mais qui sont faciles à interpréter pour des modèles spécifiques.

L'approche Late Binding est la plus utilisée pour des modèles génériques.
Par exemple, EXPRESS définit un point par trois attributs de type réel correspondants aux coordonnées x, y et z du point comme suit :

ENTITY point ;

X, Y, Z : REAL ;

END ENTITY ;

Le même point, modélisé en XML en utilisant l'approche Early Binding est :

< point id = "e1" >

<x> 4.1 </x>

<y> 2.7 </y>

<z> 1.0 </z>

< /point>

Ou l'attribut *"id"* représent un identificateur unique non modélisé.

En utilisant l'approche Late Binding, le même point est :

< entity name = "point" id = "e1" >

< attribute name = "x" >

< real value> 4.1 </real value >

< /attribute>

< attribute name = "y" >

< real-value> 2.7 </real-value >

< /attribute>

< attribute name = "z" >

< real-value> 1.0 </real-value >

< /attribute>

< /entity >

En Late Binding, les noms des composants XML ne correspondent pas directement aux types de données EXPRESS, mais plutôt à des objets de type metadonnées : entités, attributs, type de données. Une telle correspondance Late Binding EXPRESS-XML est souhaitable pour des applications qui nécessitent plusieurs modèles d'informations EXPRESS. Si l'approche Early Binding est utilisée pour une telle application, il faut définir des groupes distincts de balises XML pour chaque modèle EXPRESS.

Le Late Binding donne la possibilité d'utiliser un seul groupe de balises pour tous les modèles EXPRESS parce qu'il défini des balises correspondant à des meta-données et non à des objets définis dans le modèle.

En essayant de reformuler un modèle EXPRESS en XML, plusieurs problèmes se posent : le domaine de visibilité d'un nom, la hiérarchisation, l'agrégation, etc. Dans un modèle de données EXPRESS, le nom d'une entité doit être unique. Par contre, le nom d'un attribut ne doit être unique qu'à l'intérieur d'une entité. En même temps, des attributs avec le même nom peuvent coexister dans une entité si les types des données représentées sont différents. Mais, si le document XML qui décrira le modèle EXPRESS est muni d'un DTD, des noms d'attributs EXPRESS qui se répété ne peuvent pas correspondre directement aux éléments XML.

Barkmeyer[13] propose trois approches pour résoudre ce problème. La première approche associe les attributs EXPRESS avec les attributs d'un élément XML correspondant à l'entité. Le problème avec cette approche est l'impossibilité de représenter tous les types d'attributs EXPRESS. La représentation XML dans une telle approche sera :

< point x-id= "e1" x= "4.1" y= "2.7" z= "1.0" >

Une autre approche proposée, qui fait correspondre les attributs EXPRESS à des éléments XML, utilise des balises XML de type nom-entité.nom-attribut. Le but de ce choix est d'assurer l'unicité des noms d'attributs XML, mais ceci complique beaucoup le traitement des fichiers XML.

La représentation XML sera :

< point id= "e1" >

< point.x > 4.1 < /point.x >

< point.y > 2.7 < /point.y >

< point.z > 1.0 < /point.z >

< /point >

La troisième approche propose l'utilisation des DTD dans le contexte de early-binding pour les attributs qui peuvent être facilement définis et l'utilisation des structures comme XML Schema [?] pour les autres.

Le modèle EXPRESS n'a pas une structure hiérarchique intrinsèque. Rien dans un modèle EXPRESS n'identifie qu'une instance d'une entité fait partie d'une autre entité de plus haut niveau. Un modèle EXPRESS est donc une collection de types d'entités indépendantes liés par des relations. Les relations sont réalisées en utilisant des attributs qui ont comme valeurs des instances d'entités.

Donc, un modèle de données EXPRESS est converti en une séquence d'éléments XML qui représente des instances d'entités EXPRESS. Chaque élément-entité a un attribut de type identificateur unique XML. Donc, les attributs EXPRESS correspondent à des éléments XML et non pas à des attributs. L'élément XML, qui modélise l'entité EXPRESS, contient lui-même des éléments qui représentent les attributs EXPRESS. En plus, on fait correspondre les types d'entités en utilisant des éléments-référence qui sont des éléments ayant un attribut de type référence (IDREF).

Par exemple, une entité de type triangle est définie en EXPRESS comme suit :

```
ENTITY triangle;
    p1, p2, p3 : point;
END-ENTITY;
```

En XML, le même triangle sera :

```
<triangle id= "triangle">
    <p1></triangle-ref ref= "e1"></p1>
    <p2></triangle-ref ref= "e2"></p2>
    <p3></triangle-ref ref= "e3"></p3>
</triangle>
```

EXPRESS utilise l'agrégation pour les attributs, ce qui est modélisé en XML en utilisant une séquence d'éléments. La base est un élément XML correspondent au type de base d'agrégat.

Par exemple un polygone défini comme une séquence de trois ou plusieurs points en EXPRESS :

ENTITY polygone;
noeuds : LIST [3 :?] OF point;
END-ENTITY;

Sera représenté en XML comme :

< polygone id= "polygone" >
< noeuds >
< /point-ref ref= "e1" >
< /point-ref ref= "e2" >
< /point-ref ref= "e3" >
.....
< /noeuds >
< /polygone >

La compagnie STEP TOOLS[3] a développé une librairie nommée ST-XML pour facilité la représentation des fichiers STEP en XML. Le *binding* est composé de deux étapes. Dans la première étape, à partir du modèle de données EXPRESS, on génère le dictionnaire de données (DTD) correspondant. Dans la deuxième étape, en partant du fichier des données STEP et en se basant sur le DTD généré, on génère le fichier XML correspondant.

À la suite d'une traduction de STEP à XML, on obtient des fichiers XML beaucoup plus grands, mais beaucoup plus facile à lire et à interpréter versus des fichiers STEP plus petits - mais très difficiles à lire et à interpréter.

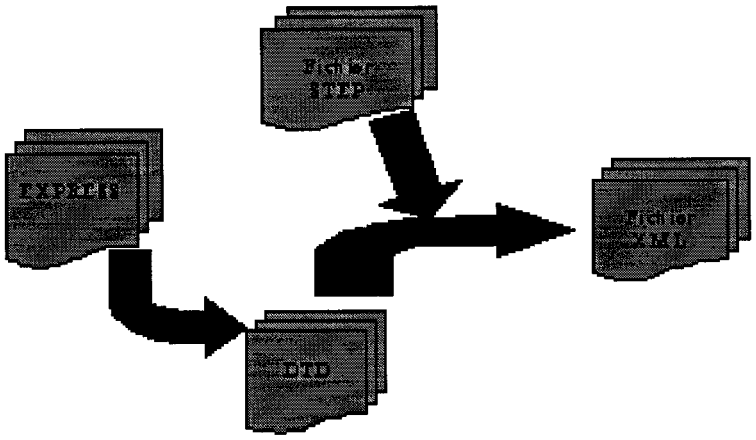


FIGURE 2.1 Le processus de *binding*.

De plus, en utilisant une grammaire (DTD ou XSD) qui défini la structure et le comportement d'un modèle de données CAO ou CFD, on peut facilement et sans perte d'informations assurer l'échange des données entre des applications.

2.6 CGNS - *CFD General Notation System*

Le projet CGNS [1] est né en 1994-1995 comme un effort concerté entre Boeing et la NASA, avec comme but de faciliter le transfert des technologies de la NASA vers des usages industriels. En 1999, le contrôle du projet CGNS a été complètement transféré à un forum public. Le comité de coordination se compose des représentants internationaux de gouvernements et de l'industrie privée. Puisque le principal empêchement aux transfert des technologies vers l'industrie était la disparité des formats d'entrée-sortie utilisés par les différents outils de simulation numérique, CGNS a été conçu pour promouvoir la CFD «*plug-and-play*» .

Le but spécifique de CGNS est de fournir une norme pour l'enregistrement et la récupération des données CFD. En conformité avec les spécifications de CGNS, le

format implémenté par la norme doit être général, portable, extensible et durable.

Étant une norme créée pour des données CFD, elle vise à stocker des solutions Navier-Stokes, la connectivité multi-zone, les coordonnées de maillage, les éléments et solutions structurées et non structurées, les conditions aux frontières, les dépendances en temps et d'autres types de données comme les états de référence et l'historique de convergence.

CGNS est une collection de conventions et l'implantation logicielle de ces conventions conçues pour enregistrer et récupérer des données CFD. Le système est composé de deux parties : l'entité conceptuelle et l'entité physique ou l'implantation logicielle.

L'entité conceptuelle représente une collection de conventions et de définitions pour l'archivage des données CFD. Ici, on trouve le «*Standard Interface Data Structures* (SIDS)» et le «*File Mapping*» qui définissent la localisation exacte d'une donnée dans un fichier.

L'entité physique, ou l'implantation, représente le produit logiciel qui, en concordance avec les concepts définis dans le SIDS, assure les opérations d'entrée/sortie.

Le SIDS constitue l'essence de CGNS. Il définit le contenu intellectuel des données CFD, l'organisation de la structure de données et les conventions de noms.

Le *File Mapping*, spécifie la méthode exacte par laquelle, en utilisant les conventions CGNS, une structure de données CFD sera stockée dans un fichier binaire ADF.

Le Format de Données Avancé (*Advanced Data Format*-ADF) est un ensemble de routines logicielles qui réalisent les opérations d'entrée/sortie sur une base de données (fichiers binaires ADF).

La librairie *Mid-Level* est une interface d'application (API) qui facilite l'accès de haut niveau aux données stockées dans une base de données ADF. Elle est la partie

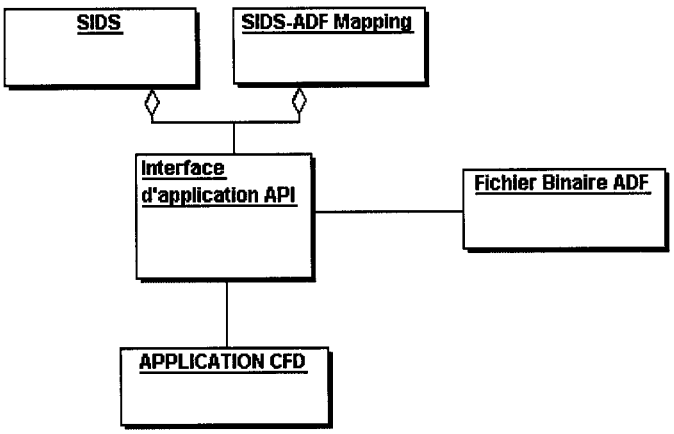


FIGURE 2.2 La philosophie du CGNS

la plus visible aux utilisateurs, et écrite en ANSI C pour assurer la compatibilité.

Parmi les caractéristiques de l'ADF, on peut énumérer : la structure de données hiérarchique basée sur une seule structure de données nommée noeud ADF, qui est un graphe dirigé et qui peut supporter n'importe quel type de données (pas seulement CFD). Le logiciel est écrit en ANSI C pour lui assurer la portabilité et les données sont stockées en format C binaire.

Un tel noeud ADF a la structure suivante :

- ID : un identificateur unique pour accéder à un noeud dans un fichier
- Name : chaîne de caractères utilisée pour nommer un noeud
- Label : chaîne de caractères utilisée pour décrire le type d'information contenue dans le noeud
- Data Type : chaîne de caractères spécifiant le type de donnée (réal, int, complexe, etc.) associé à chaque noeud
- Dimensions : un vecteur entier qui contient le nombre d'éléments pour chaque dimension

- Data : la donnée associée au noeud
- Number of sub-nodes : nombre d'enfants associés au noeud
- Name of sub-nodes : la liste des noms des enfants

La base de données est hiérarchique, chaque fichier de type ADF ayant une racine auquel les noeuds de type CGNSBase_t sont liées. Une suite de zones, c'est-à-dire des groupes d'éléments de discrétisation, leurs coordonnées, les solutions, leurs connectivités, sont liés à ce noeud de base de type CGNSBase_t. Ensuite, d'autres noeuds comme FlowEquationSet_t, Family_t ou Description_t, peuvent être liés au noeud CGNSBase_t.

La norme CGNS, supporte des topologies de maillage structurées et non-stucturées en utilisant la même structure de données Zone_t. Deux noeuds font la différence entre ces deux topologies :ZoneType_t (Structured, Unstructured) et Elements_t (ElementsConnectivity, shape, range, etc.). Aussi, de nouveaux paramètres étaient nécessaires pour concilier les différences entre les éléments structurés et les éléments non-structurés : *IndexDimension*, qui représente le nombre de coordonnées de calcul nécessaires pour définir de façon unique un noeud dans un maillage, *CellDimension* qui représente la dimension des cellules du maillage (1D, 2D, 3D) et *PhysicalDimension*, qui définit la dimension physique de l'espace.
Le tableau 2.1 montre comment on peut définir tous les types de maillages à l'aide de ces paramètres. La norme CGNS n'est pas conçue pour garder des géométries[17]. Les concepteurs de CGNS ont décidé de ne pas utiliser un format spécifique de stockage de la géométrie mais plutôt d'utiliser des liens vers des formats CAO existants comme IGES, STEP, Pro-Engineer, etc.

L'association maillage-géométrie étant nécessaire pour raccourcir le temps de réponse aux changements dans les designs, pour l'adaptation de maillage ou pour l'analyse et l'affichage des résultats, cette information a été ajoutée dans la structure, en utilisant une couche d'indirection. La solution utilisant une couche d'indi-

TABLEAU 2.1 Les paramètres utilisés par CGNS pour définir un maillage

Les types de maillage	IndexD	CellD	PhysD
3D structuré	3	3	3
2D structuré en espace 2D	2	2	2
2D structuré en espace 3D	2	2	3
1D structuré en espace 1D	1	1	1
1D structuré en espace 2D	1	1	2
1D structuré en espace 3D	1	1	3
3D non-structuré	1	3	3
2D non-structuré en espace 2D	1	2	2
2D non-structuré en espace 3D	1	2	3
2D non-structuré en espace 1D	1	1	1
2D non-structuré en espace 2D	1	1	2
2D non-structuré en espace 3D	1	1	3

rection a été adoptée parce qu'il y avait rarement une connectivité 1 à 1 entre les régions de maillage et les entités géométriques. L'utilisation de cette couche d'indirection donne la possibilité de créer des associations insensibles aux changements du maillage et de la géométrie et en même temps de définir les conditions frontières et des propriétés de matériaux sur cette couche. CGNS définit donc l'association entre le maillage et la géométrie en référant à des entités CAO directement dans leurs fichiers. L'implémentation physique est réalisée en utilisant une structure de données nommée *Family_t*.

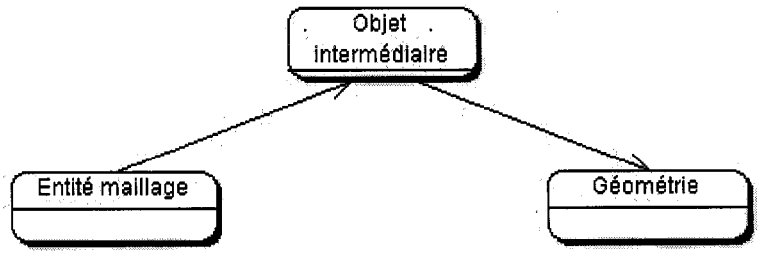


FIGURE 2.3 L'objet intermédiaire en CGNS

La structure de données "*Family_t*" contient le nom du fichier CAO où la géométrie

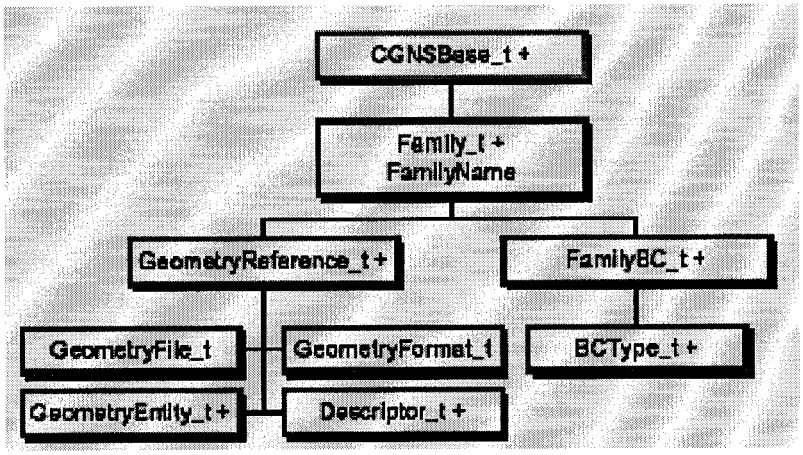


FIGURE 2.4 La structure de données *Family* du CGNS

est stocké (*GeometryFile_t*), le type de format de géométrie (*GeometryFormat_t*) et le nom d'entité géométrique (*GeometryEntity_t*). Si le nom d'entité géométrique n'est pas spécifié, on considère par défaut que le nom de famille et le nom d'entité sont les mêmes.

Dans un même fichier de données de type ADF, peuvent coexister plusieurs bases de données CGNS, chacune ayant un noeud de type *CGNSBase_t* d'un nom différent. Du moins théoriquement, pour éviter la redondance de données, par exemple plusieurs groupes de solution sur le même maillage - la norme dispose d'un mécanisme capable de garder des liens vers d'autres bases de données ADF. En pratique ce mécanisme de référence ne fonctionne pas encore à cent pourcent.

Les autres structures de données qui peuvent être stockées par CGNS sont les suivantes : *RigidGridMotion_t*, qui enregistre les translations et rotations rigides du maillage, *ArbitraryGridMotion_t*, qui enregistre point par point les mouvements ou les déformations du maillage et *IterativeOrTemporalData_t*, structure qui associe à chaque pas de temps ou itération une structure de données appropriée pour le mouvement rigide et arbitraire du maillage, pour les coordonnées du maillage ou pour les solutions.

Étant donné la généralité qu'elle fournit pour la représentation de l'information numérique CFD et son habilité à garder des liens vers des géométries, la norme CGNS a été retenue pour bâtir le modèle de données CFD de ce projet.

2.7 Autres Standards

2.7.1 PIRATE

Développé au Centre de recherche en calcul appliqué (CERCA), PIRATE est une bibliothèque de classes écrite en C++ pour la conception de logiciels de simulation. La bibliothèque PIRATE propose un noyau de classes d'objets, riche et extensible, permettant de représenter et de manipuler les données utilisées non seulement à l'intérieur du solveur (variables, conditions frontières, opérateurs différentiels, etc.), mais également durant les phases de préparation des données (géométrie, maillage, fonctions de concentration, etc.) et d'analyse (solution, estimé d'erreur, etc.). La bibliothèque comprend également des outils de gestion de ces données qui permettent entre autre de sauvegarder sur disque ou de recharger toutes les données pertinentes à une simulation ou d'échanger ces données entre deux modules durant le processus de résolution. La bibliothèque définit et utilise un format d'échange des données entre des différentes applications (le format .pie).

Un fichier pie valide est constitué d'un ou plusieurs des éléments suivants : *champ, géométrie, maillage et solutions.* Chaque élément étant optionnel. Le bloc géométrie regroupe les éléments géométriques (point, courbe, surface), topologiques (sommets, arrête, face, volume), solides (sphère, cylindre) ou visuels (annotation). Le bloc maillage regroupe une suite de zones, c'est-à-dire des groupes d'éléments géométriques de discrétisation et leurs coordonnées. Chaque zone est identifiée par un

identificateur unique. PIRATE peut travailler avec des zones structurées et non structurées, les deux types de zones pouvant coexister dans la même structure.

Le bloc solution regroupe une suite de variables, c'est-à-dire des groupes d'éléments de calcul et les valeurs des variables. Un champ correspond à un tableau de valeurs. Il est utilisé, par exemple, pour spécifier les coordonnées des noeuds du maillage, sa connectivité ou encore les valeurs d'une variable de la solution. Chaque champ possède un identificateur différent, Id, qui servira à établir des références à ce champ. Un objet de type champref est utilisé pour indiquer une référence à un champ dans la définition des autres éléments du fichier. Généralement, les valeurs sont définies dans des champs et leurs utilisation est indiquée par un champref. En plus de l'identification du champ à utiliser, un champref définit aussi comment accéder à l'information.
Un exemple d'une fichier de données PIRATE est montré dans l'annexe II.

2.7.2 VRML Virtual Reality Modeling Language

VRML est un standard ISO[4], crée en 1994 pour la représentation 3D sur le Web, qui donne une plate-forme versatile, pour une variété d'applications qui utilisent le 3D. Un de ses points forts est la facilité d'interagir avec d'autres standards.
Un fichier VRML est formé d'un ensemble de noeuds qui définissent le contexte et le monde virtuel, dans lequel on trouve des éléments de spécifications : des formes géométriques de base (cube, cône, sphère), d'objets à facettes polygonales, l'aspect (couleur, brillance, etc.), la texture appliquée sur les objets, des groupes d'objets, des transformations, le positionnement et les transformations (translation, rotation, mise à l'échelle).

2.7.3 XVL - eXtensible Virtual world description Language

En essayant d'intégrer VRML et XML, un autre standard est né : eXtensible Virtual world description Language [6] . Celui-ci est un langage qui fait partie de VRML, mais qui a des propriétés très intéressantes : il peut créer des fichiers très petits, 1% de la dimension d'un fichier VRML, ce qui le rend idéal pour la transmission sur Internet. XVL support aussi la modélisation des solides, il peut représenter des formes de surfaces précises et bien sûr il peut supporter l'animation interactive basée sur des scripts.

En faisant le bilan des approches existantes et en considérant leur maturité pour décrire et manipuler des données d'analyse numérique, nous avons choisi CGNS et XML comme étant les formats les plus appropriés.

CHAPITRE 3

DESCRIPTION DE LA STRUCTURE DE DONNÉES DU MODÈLE OPALE BASÉE SUR LA NORME CGNS

L'objectif principal de ce projet est de développer un modèle de représentation de données pour le processus d'analyse et d'optimisation, basé sur des formats de description des données ouverts et matures. Le format de représentation de données doit être assez général et mature pour pouvoir garder toutes les informations nécessaires dans un processus d'analyse numérique.
De plus, le modèle de données doit satisfaire les besoins spécifiques du projet OPALE d'automatisation du processus d'optimisation du design des pales des turbines hydrauliques. Plus précisément, le but de ce modèle est :

- de donner un cadre qui facilite l'intégration d'applications («maison» et commerciales) d'analyse et de design,
- de donner la possibilité de considérer les activités de design dans un cycle itératif,
- de donner la possibilité d'échanger des données CFD.

La description détaillée du processus d'optimisation basé sur le modèle de données choisi sera présentée dans le chapitre suivant.

Cette présentation du modèle n'a pas pour but de répéter les informations du document SIDS (Standard Interface Data Structure), qui est le document officiel de description du standard CGNS. Nous nous proposons plutôt de décrire du point de vue conceptuel le modèle de données OPALE, basé sur la norme CGNS.
La tâche reliée à l'implémentation réelle du modèle est présentée dans un autre chapitre. Ce chapitre documente les contraintes dans l'utilisation de CGNS dans le cadre du projet OPALE, les problèmes d'interprétation de la norme CGNS et les

choix particuliers qui ont été faits.

3.1 Spécifications des noeuds

Ce chapitre présente chaque entité qui compose le modèle Opale du point de vue des contraintes et des particularités de ce projet. La structure de données étant arborescente, nous avons choisi de suivre cette arborescence pour décrire le modèle. Chaque niveau de la structure de données est représenté graphiquement par les noeuds requis à ce niveau. Chaque entité du graphique est décrite par deux paramètres : le nom de l'entité identifié par «*Name*» et le type d'instance identifié par «*Label*». Les deux ont été choisis en conformité avec le SIDS. De plus, nous avons choisi de décrire chaque entité à l'aide de l'ensemble de propriétés suivantes :

Description : brève description du noeud (telle que dans le SIDS)

Parent(s) : les parents possibles du noeud

Enfants :

Enfants requis : la liste des enfants requis par Opale

Enfants optionnels : la liste des enfants optionnels

Données : les données stockées dans le noeud.

Chaque noeud enfant est suivi par un coefficient de multiplicité. La convention utilisée est la suivante :

NomNoeud [coefficient] ou coefficient est l'un des symboles :

"+" signifie qu'une ou plusieurs instances de ce type sont autorisées

" ?" signifie que zéro ou une seule instance de ce type est autorisée

"*" signifie que zéro ou plusieurs instances de ce type sont autorisées

"#" signifie qu'une et seulement une instance de ce type est autorisée

3.2 Le noeud CGNSBase

CGNSBase_t est le noeud de plus haut niveau d'une base de données CGNS. Il permet de regrouper toutes les entités nécessaires pour décrire au complet les solutions d'un problème CFD. CGNS prévoit la possibilité que plusieurs bases de données coexistent dans un même fichier, donc plusieurs noeuds de ce type dans un même fichier. Nous n'autorisons pas cela ici, à cause des différents problèmes qui peuvent en découler, la norme CGNS ne disant rien concernant la façon de traiter les données et sur l'interprétation de chaque base. Pour pouvoir décrire au complet un problème dans le cas des turbines hydrauliques, nous avons établi que les entités suivantes sont requises :

DataClass #
DimensionalUnits #
ReferenceState #
FlowEquationSet #
Zone +
Gravity #
Family #
Descriptor #
ConvergenceHistory #
SimulationType #

Ces noeuds sont présentés dans les sous-sections et sections ultérieures.

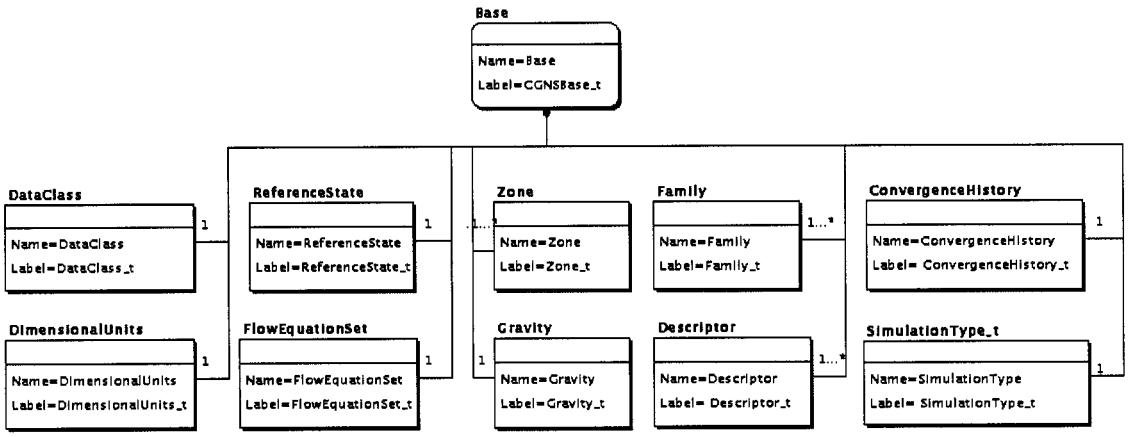


FIGURE 3.1 Le noeud CGNSBase

CGNSBase	
Description	Le noeud de plus haut niveau d'une base de données CGNS
Parent(s)	aucun
Enfants requis par Opale	- DataClass #, DimensionalUnits # - ReferenceState # - FlowEquationSet # - Zone + - Gravity # - Family # - Descriptor # - ConvergenceHistory # - SimulationType #
Enfants optionnels	aucun
Données	cell_dimension et physical_dimension

3.2.1 Les noeuds DataClass et DimensionalUnits

CGNS donne une certaine liberté pour définir si les unités sont dimensionnelles ou non tout en respectant la règle de la précédence dans la hiérarchie [voir SIDS section 4]. Nous avons décidé d'imposer certaines restrictions à ce niveau dans la structure de données, en imposant l'existence de ces informations au niveau de la base. Cela ne signifie pas qu'il n'y a pas la possibilité de redéfinir d'autres unités à un autre niveau de la structure, mais la règle de précédence fait que cela assure un système d'unités par défaut. Pour pouvoir décrire la classe de données et toutes les autres informations dimensionnelles nécessaires pour la manipulation des données, nous imposons comme requis les noeuds DataClass et DimensionalUnits au niveau du noeud CGNSBase. DataClass est un type énumératif qui classe les données en catégories dépendant des dimensions et de la normalisation des données associées. Dans le cas d'Opale, DataClass est fixée à «*Dimensional*» ce qui signifie que toutes les données sont stockées avec des unités. L'entité structurale DimensionalUnits décrit le système d'unités utilisé pour mesurer les données ayant des dimensions.

3.2.2 Le noeud ReferenceState

Dans le modèle Opale, des données numériques invariables, comme la température et la densité, sont requises. La température et la densité sont constantes, car Opale ne traite que les écoulements incompressibles. Elles doivent être stockées comme des informations globales au niveau de la base. CGNS peut stocker ces informations comme des références globales au niveau du noeud ReferenceState, qui décrit l'état de référence par une liste (regroupement) de conditions communes. Les autres paramètres qui seront stockés ici sont ceux qui déterminent la viscosité. Les quatre paramètres (Reynolds, Reynolds_Velocity, Reynolds_Length, Reynolds_ViscosityKinematic) nécessaires pour la définir sont requis dans la structure du noeud ReferenceState.

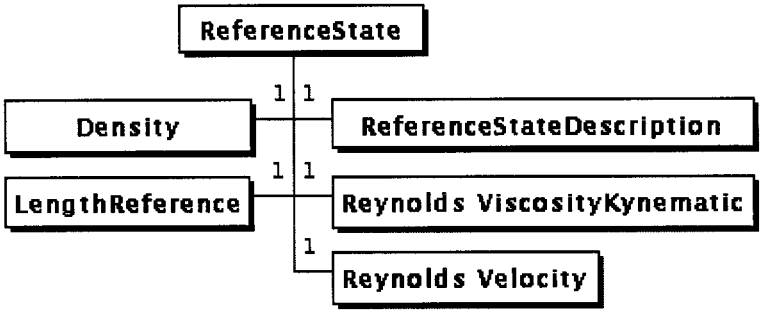


FIGURE 3.2 ReferenceState

Le noeud ReferenceState	
Description	le noeud qui regroupe tous les paramètres de calcul
Parent(s)	CGNSBase
Enfants requis par Opale	- ReferenceStateDescription # -Reynolds_ViscosityKinematic # - Reynolds_Velocity # - Density #
Enfants optionnels	- Descriptor
Données	aucune

3.2.3 Le noeud ReferenceStateDescription

Nous imposons comme requis le noeud ReferenceStateDescription au niveau du noeud ReferenceState. C'est une façon de rendre plus claire les données stockées dans la base de données CGNS dédiée à l'optimisation des turbines hydrauliques. Ce noeud ne fait que documenter de manière informelle les conditions initiales imposées par le noeud ReferenceState. Dans le cas d'Opale, le noeud ReferenceState est inséré dans la base de données lors de la standardisation de la base. Le noeud ReferenceStateDescription est inséré au même moment et contient par défaut le nom de fichier XML utilisé pour générer le noeud ReferenceState.

3.2.4 Le noeud FlowEquationSet

La description des équations utilisées pour accomplir la tâche d'analyse numérique est une autre donnée importante dans le processus d'analyse. Cela nous a amené à considérer comme obligatoire l'existence de ces informations au niveau de la base. Pour stocker ces informations, CGNS met à la disposition l'entité FlowEquation-Set qui, conformément aux spécifications du SIDS, peut exister comme enfant du noeud CGNSBase, du noeud Zone ou des deux. Dans le contexte spécifique de la

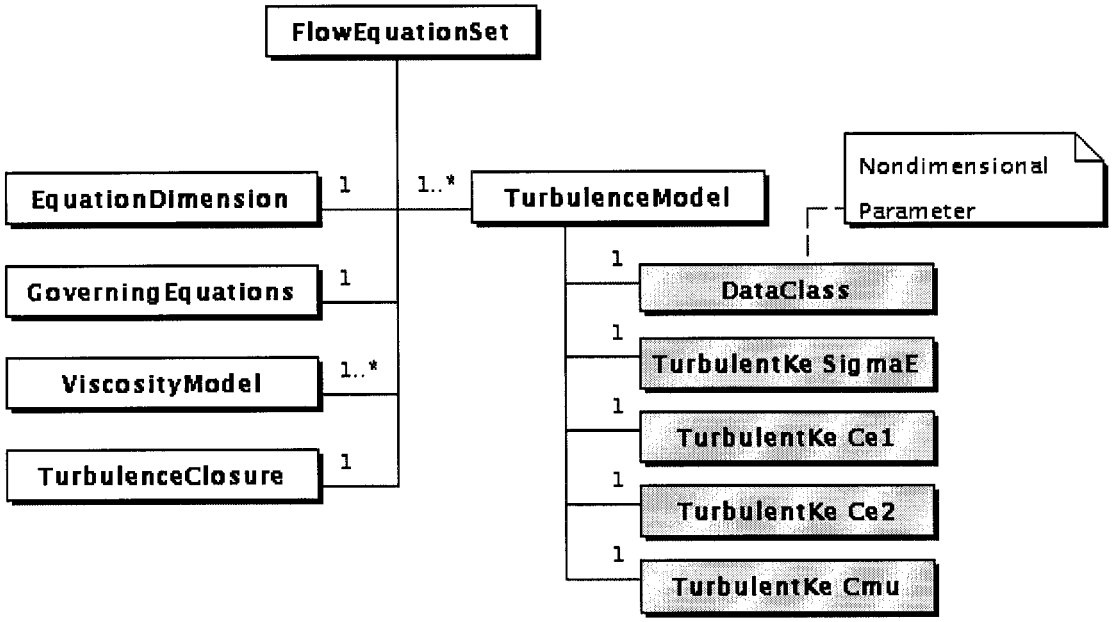


FIGURE 3.3 FlowEquationSet

simulation numérique des turbines hydrauliques, l'ensemble des équations utilisées est toujours le même, car nous n'avons pas de calcul de structure mais uniquement des calculs de fluides. Cela nous a amené à imposer l'existence du noeud FlowE-quationSet comme enfant du noeud CGNSBase. Plusieurs paramètres requis pour définir les équations sont invariants dans ce modèle, ce qui nous a amené à imposer l'existence des noeuds appropriés pour les décrire, même si la norme n'impose pas leur existence. Ces paramètres sont passés en revue à la section 3.3.

Le noeud FlowEquationSet	
Description	Structure qui permet de décrire l'ensemble des équations utilisées dans le processus de simulation numérique
Parent(s)	- CGNSBase
Enfants requis par Opale	- EquationDimension # - GoverningEquations # - ViscosityModel + - TurbulenceClosure # - TurbulenceModel + - DataClass # - TurbulentKe_SigmaE # - TurbulentKe_Ce1 # - TurbulentKe_Ce2 # - TurbulentKe_Cmu #
Enfants optionnels	- DataClass - DimensionalUnits - DimensionalExponents
Données	- aucune

3.3 Les paramètres généraux

Un des paramètres requis par le modèle est la dimension des équations, qui est défini par le noeud EquationDimension. Sa valeur est fixée à 3. Le type d'équations utilisées dans le cas des turbines hydrauliques est toujours NSTurbulentIncompressible et l'information est stockée comme donnée au niveau du noeud GoverningEquations.

3.3.1 Les paramètres de la viscosité

Un autre paramètre requis par le modèle est la viscosité moléculaire, qui est constante dans le cas de l'hydraulique. Cela nous a amené à imposer l'existence

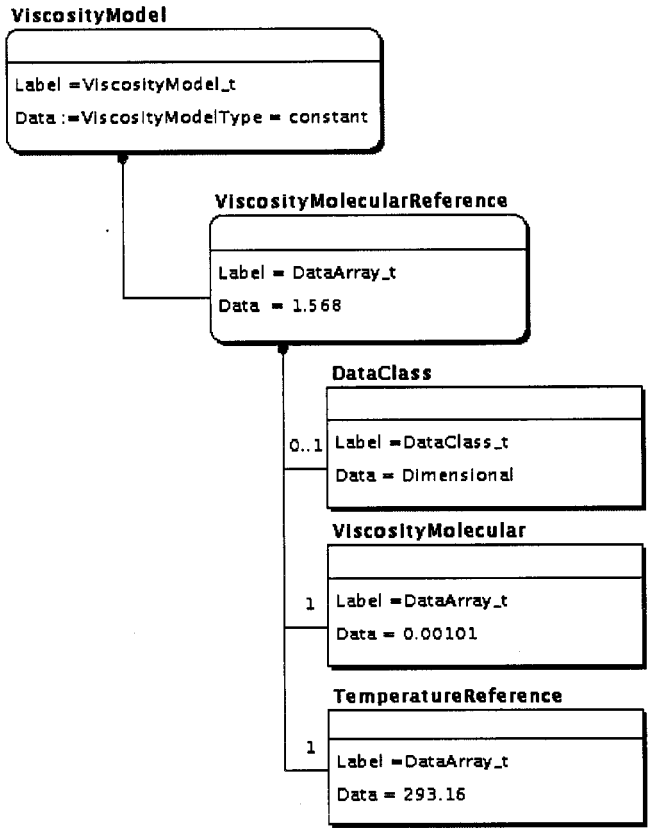


FIGURE 3.4 Viscosity

du noeud ViscosityModel comme enfant requis du noeud FlowEquationSet. Pour décrire la viscosité moléculaire, nous avons besoin de connaître le type de viscosité, la valeur de la viscosité et la température de référence. Le paramètre décrivant le type de viscosité «*ViscosityModelType*»est stocké au niveau du noeud ViscosityModel. Dans notre cas, il prend toujours la valeur «*constant*». La valeur de la viscosité moléculaire et la température de référence étant des paramètres requis, nous avons choisi d'imposer l'existence de deux noeuds de type DataArray ayant les

noms ViscosityMolecular et TemperatureReference comme enfants du noeud ViscosityMolecularReference.

3.3.2 Les paramètres de la turbulence

Pour fin de référence il est utile de conserver tous les paramètres des équations de Navier-Stokes utilisés dans le processus d'optimisation des turbines hydrauliques, d'où la nécessité de stocker ces paramètres dans la base de données à l'aide d'une structure définissant le modèle de turbulence. Il est possible de le définir en CGNS en utilisant l'ensemble de noeuds spécialisés TurbulenceClosure et TurbulenceModel. Le seul paramètre requis par le noeud TurbulenceClosure est le type de modèle de turbulence, qui, dans notre cas est toujours «*EddyViscosity*». Le paramètre qui décrit l'ensemble des équations utilisées pour modéliser la turbulence, «*TwoEquation_LaunderJones*» dans notre cas, est stocké en utilisant la structure TurbulenceModel. Nous avons imposé les noeuds DiffusionModel et DataClass comme requis dans la structure de données TurbulenceModel. En utilisant la même structure, nous gardons la liste des constantes associées à ce modèle qui sont : TurbulentKe_SigmaK, TurbulentKe_SigmaE, TurbulentKe_Ce1, TurbulentKe_Ce2, TurbulentKe_SigmaK et TurbulentKe_Cmu.

3.4 Le noeud Zone

Le noeud Zone représente une structure de données qui regroupe toutes les informations pertinentes concernant un sous-domaine. La norme CGNS donne toutes les structures de données requises (par le noeud Zone) pour définir au complet un sous-domaine, mais elle n'impose pas que le sous-domaine soit complètement défini. Pour les besoins spécifiques à ce projet, nous voulons qu'un sous-domaine soit

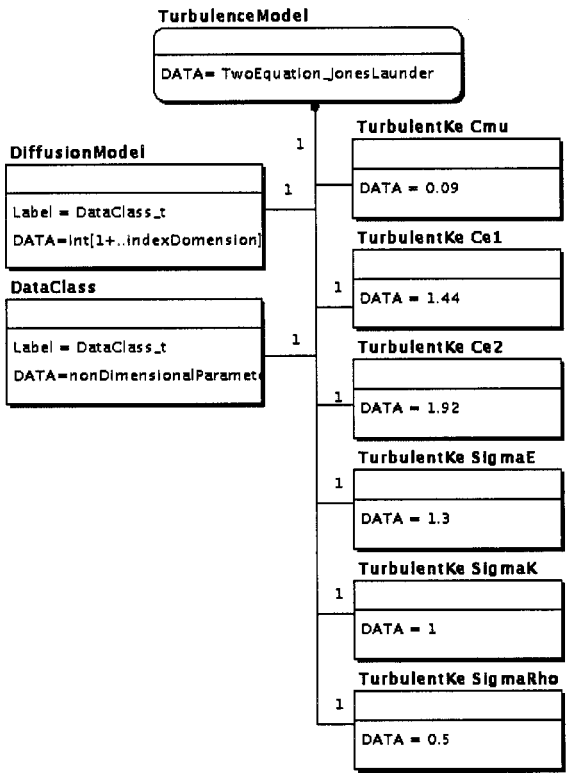


FIGURE 3.5 Structure utilisée pour décrire la turbulence

complètement défini, ce qui nous a conduit à imposer le minimum de noeuds requis pour pouvoir décrire un sous-domaine. Le nombre et les types des noeuds requis pour décrire une zone varient en fonction du type de maillage et du nombre de sous-domaines stockés dans la base de données. S'il s'agit d'un maillage non-structuré, les cinq structures de données suivantes sont requises pour définir un maillage non-structuré et les solutions calculées dessus :

ZoneType #

GridCoordinates #

FlowSolution +

ZoneBC #

Elements #

Ce que la norme impose comme noeud requis est le type de maillage utilisé. Cela est implanté en CGNS à l'aide de l'entité ZoneType qui peut prendre soit la valeur «*structured*», soit la valeur «*unstructured*», soit une valeur définie par l'utilisateur.

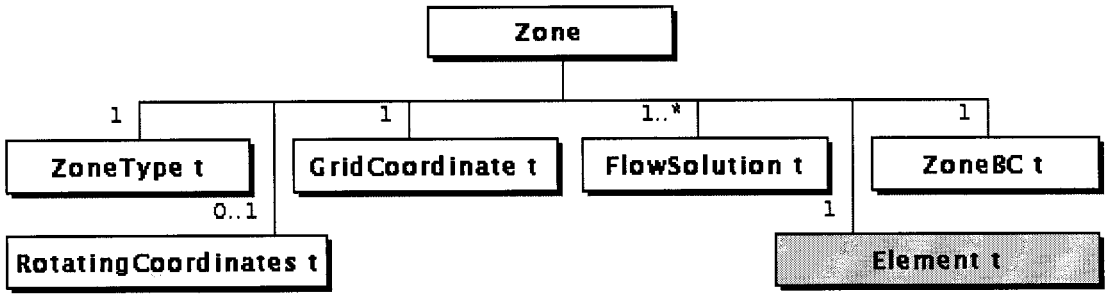


FIGURE 3.6 Structure de données définissant une zone non-structurée

Conformément au SIDS, une zone non-structurée peut contenir des éléments mixtes. La structure de données utilisée pour définir un maillage non-structuré ayant ou non des éléments mixtes est Elements. Plusieurs instances de type Element_t sont autorisées dans une zone de type non-stucturé. La structure du noeud Elements contient quatre noeuds que l'on a imposés comme requis :

IndexRange #

ElementSizeBoundary #

ElementType #

ElementConnectivity #

Dans le cas des éléments mixtes, le noeud ElementConnectivity qui définit la connectivité des éléments, contient un entier de plus par élément pour définir son type [Voir SIDS page 59]. Dans le cas d'un maillage structuré, le nombre des noeuds requis est réduit à cinq, car le noeud Elements n'est plus autorisé ici.

S'il y a plusieurs domaines (zones) de calcul dans une base de données, il faut spécifier la connectivité entre ces domaines, donc chaque structure de données «*Zone*», contiendra une entité *ZoneGridConnectivity* qui décrit la connectivité entre des

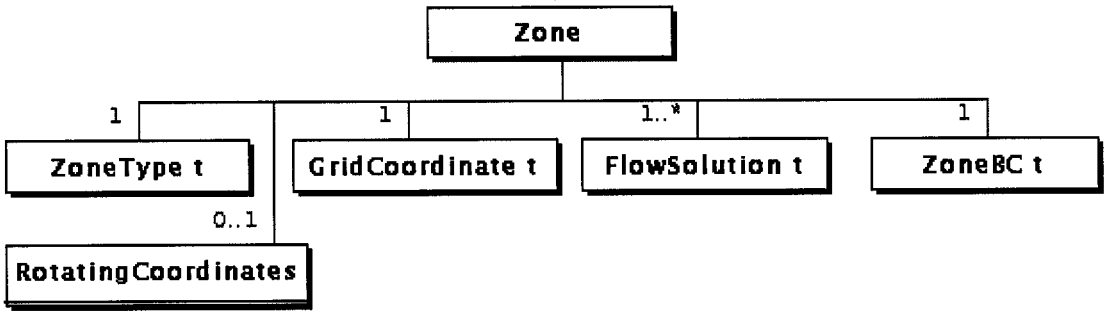


FIGURE 3.7 Structure de données définissant une zone structurée

zones. Le noeud RotatingCoordinates définit un repère en rotation et il peut exister soit au niveau de la base, soit dans une Zone. S'il est au niveau de la base, ça signifie que toute la géométrie tourne. En général, on voudra donc plutôt imposer ce noeud au niveau de la zone, dans chacune des zones en rotation.

Le noeud Zone	
Description	Le noeud qui regroupe toutes les informations pertinentes concernant un sous-domaine de calcul
Parent(s)	- CGNSBase
Enfants requis par Opale	- ZoneType # - GridCoordinates # - ZoneBC + - FlowSolutions + - Elements +
Enfants optionnels	- RotatingCoordinates - ZoneGridConnectivity - Family - Descriptor - DataConversion
Données	- aucune

3.5 Le noeud RotatingCoordinates

Le noeud RotatingCoordinates est une structure de données utilisée pour définir le centre et le vecteur de rotation d'un système de coordonnées en rotation. La structure de données ne traite que le cas ou les interfaces entre les zones sont perpendiculaires aux axes des rotation. Les deux enfants requis par ce noeud sont : RotationCenter et RotationRateVector. La présence d'un noeud RotatingCoordinates dans une Zone signifie que le calcul devrait être fait en repère tournant. Logiquement, on s'attendrait à ce que les conditions aux limites et la solution soit alors données dans le repère tournant. En particulier, le FlowSolution devrait alors avoir un noeud RotatingVelocity, qui n'a pas de sens dans une Zone qui n'a pas de repère en rotation.

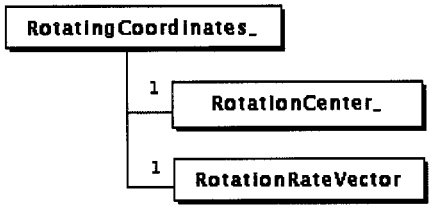


FIGURE 3.8 Structure de données dèfinisant les coordonnées de rotation

Le noeud RotatingCoordinates	
Description	le noeud qui stocke les coordonnées d'un système en rotation
Parent(s)	Zone
Enfants requis par Opale	- RotationCenter # - RotationRateVector #
Enfants optionnels	- DataClass - DimensionalUnits
Données	Les coordonnées du centre de rotation et le vecteur de rotation

3.6 Le noeud GridCoordinates

GridCoordinates représente la structure de données qui décrit les coordonnées physiques du maillage [voir le SIDS, section 7]. La norme CGNS donne la possibilité de décrire un maillage à l'aide de plusieurs systèmes de coordonnées (cartésien, cylindrique, sphérique et auxiliaire). De plus, la norme donne la possibilité de spécifier la dimension de données de façon indépendante pour chaque coordonnée qui décrit le système choisi. Par exemple, conformément au SIDS, un maillage 3D défini par un système de coordonnées cartésiennes (CoordinateX, CoordinateY et CoordinateZ), peut avoir des systèmes d'unités différentes pour les trois axes. Pour spécifier la non-homogénéité, on utilise l'entité DataClass pour spécifier le système d'unités pour chaque axe. Si le système d'unités utilisé est le même pour les trois axes, mais diffère du cas par défaut, il suffit de définir les paramètres une seule fois au niveau de l'entité GridCoordinates. Étant donné la confusion et les erreurs que pourrait entraîner l'utilisation de ces possibilités, nous avons décidé d'imposer que les coordonnés soit définies en utilisant le même système d'unités pour tous les axes. En plus nous n'acceptons que des coordonnées cartésiennes et cylindriques.
Un autre champ de données optionnel, mais autorisé par la norme CGNS au niveau de GridConnectivity, est le noeud «*Rind*». Une zone peut contenir des données de type maillage ou solutions définies sur un ensemble de points en dehors de la zone elle-même. Ces points qui sont des sommets ou des cellules fictives et qui forment une peau autour d'une zone, s'appellent des «*rind*». La structure «*rind*» décrit le nombre de plans (couches) de «*peau*» associés à une zone structurée. La structure «*RindPlanes*» contient l'indice correspondant à chaque face, qui en 3D est :

n = 1 -> i-min

n = 2 -> i-max

n = 3 -> j-min

n = 4 -> j-max

n = 5 -> k-min

n = 6 -> k-max

Pour un maillage 3D structuré decrit par II x JJ x KK, une valeur de RindPlanes = [a, b, c, d, e, f] signifie que le maillage avec la peau sera décrit par les indices :

i : (1 - a, II + b)

j : (1 - c, JJ + d)

k : (1 - e, KK + f)

Comme nous n'utilisons pas de «rinds», nous avons décidé de ne pas les autoriser dans le modèle Opale.

3.6.1 Le noeud ZoneGridConnectivity ou l'interface de connectivité entre zones

Dans le cas où plusieurs sous-domaines sont définis dans une même base de données, une structure qui définit la connectivité entre des zones est requise. CGNS utilise un concept qui s'appelle interface de connectivité multi-zone (Multizone Interface Connectivity) et qui est implantée à l'aide de l'entité ZoneGridConnectivity [Voir le SIDS section 8]. Le concept utilisé par CGNS est de dupliquer les informations concernant la connectivité pour les zones adjacentes. Autrement dit, chaque zone contient la liste de noeuds communs avec les zones qui lui sont adjacentes. Chaque liste est implanté à l'aide d'entitées spécialisées. Le noeud ZoneGridConnectivity dispose de trois types des structures pour implanter la connectivité :

GridConnectivity1to1 qui s'applique pour interfacer des millages (zones) structurés avec des régions topologiquement rectangulaires ;

GridConnectivity qui est le cas général et qui peut être utilisé pour interfacer des zones structurées avec des non-structurées ;

OversetHoles pour des zones superposées avec des trous.

3.6.2 Le noeud GridConnectivity1to1

La structure GridConnectivity1to1 s'applique uniquement aux zones structurées ayant une source et une destination structurées et dont l'interface peut être définie comme une région logiquement rectangulaire. Cette structure contient l'information de connectivité pour un «*patch*» d'interface avec une correspondance 1 à 1 des noeuds. Un «*patch*» est une partie d'une zone qui touche une et seulement une autre zone. Cette structure identifie le nom des zones qui sont connectées. L'indice des noeuds pour les deux zones adjacentes qui composent l'interface et l'indice de transformation d'une zone à l'autre. Si une face d'une zone touche plusieurs autres zones, plusieurs instances de la structure de données GridConnectivity1to1 doivent être incluses dans la zone.

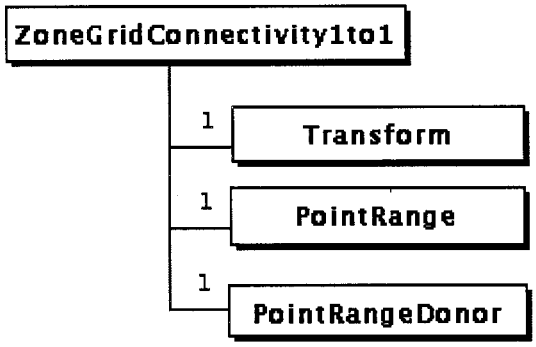


FIGURE 3.9 Structure de données définissant la connectivité 1 à 1

3.6.3 Le noeud GridConnectivity

La structure GridConnectivity contient l'information de connectivité pour les interfaces multizones généralisées, et peut être employée pour définir la connectivité entre n'importe quel type de zone structurées ou non structurée. Son but est de décrire un voisinage qui ne s'apparie pas exactement ou des interfaces qui se chevauchent. Cette structure peut également être employée pour les interfaces de voisinage 1-à-1. Pour

les interfaces de voisinage qui ne sont pas 1-à-1, également désignés sous le nom de raccordés ou de mal appariées, un «*patch*» d'interface est constitué du sous-domaine de la face d'une zone qui touche une et seulement une autre zone. Cette structure, GridConnectivity identifie a l'aide d'index le sous-domaine qui défini l'interface et donne son image dans la zone (donneur) adjacente. Elle identifie également le nom de la zone adjacente. Si une face d'une zone touche plusieurs zones adjacentes, alors plusieurs instances de type GridConnectivity sont nécessaires pour décrire toutes les interfaces. Pour une interface simple entre deux zones, deux instances de type GridConnectivity sont présentes dans la base de données - une pour chaque zone adjacente.

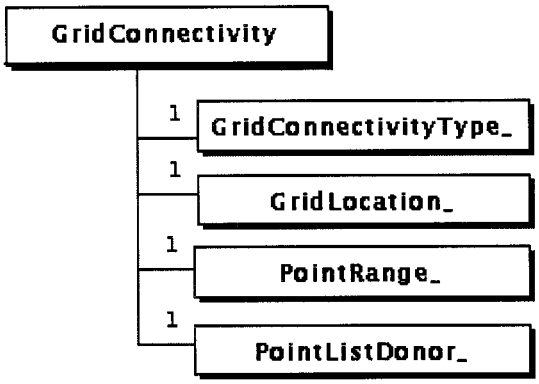


FIGURE 3.10 Structure de données définissant la connectivité générale

3.6.3.1 La périodicité

Dans le cas où le modèle contient des zones dont certaines frontières sont périodiques, la connectivité doit être définie à l'aide d'une structure de données qui permet de conserver les informations sur la périodicité. Le noeud GridConnectivity peut contenir une structure de données GridConnectivityProperty qui peut être utilisée entre autre pour définir la périodicité, à l'aide d'un noeud enfant Periodic. La structure de données Periodic contient trois noeuds de type DataArray requis par la norme

CGNS : RotationCenter, RotationAngle et Translation.

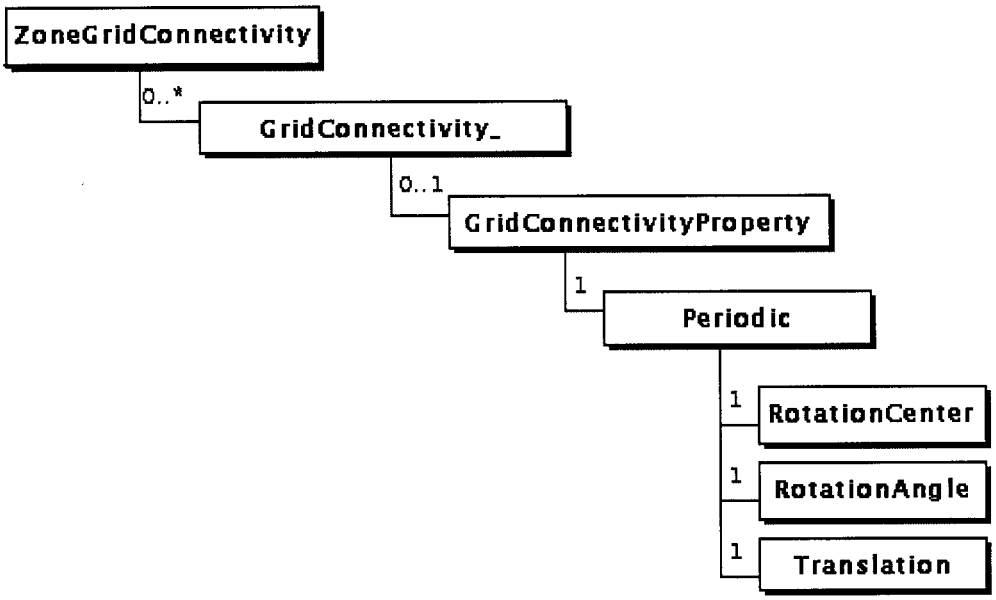


FIGURE 3.11 Structure de données définissant la périodicité

RotationCenter définit le centre de rotation, tandis que RotationAngle définit l'angle entre l'interface courante et l'interface à laquelle elle est connecté. Translation définit la translation de l'interface courante vers celle à laquelle elle est connectée.

3.7 Le noeud FlowSolution

Le noeud FlowSolution est utilisé pour stocker les résultats de simulations numériques. Selon la norme CGNS, il n'y a pas de requis concernant cette entité. La norme CGNS limite la façon de stocker des solutions calculées sur un maillage en imposant qu'elles soient toutes exprimées soit sur des sommets, soit au centre des cellules. Toutes les solutions décrites par une instance du noeud FlowSolution doivent être représentées au même endroit.

Parce qu'il y a toujours un ensemble de données solutions que nous utilisons pour

accomplir la tâche d'optimisation, nous avons imposé que ces informations soient présentes dans la base de données. Les entités requises par Opale sont :

GridLocation #

Pressure #

RotatingVelocity #

TurbulentEnergyKinetic #

TurbulentDissipation #

Nous avons imposé que l'information concernant la localisation des solutions sur le maillage soit explicitement présente dans la base de données. Le SIDS n'impose pas l'existence de ce type d'information dans la base de données. Il considère que l'absence d'une structure des données (ou d'une donnée) de la base de données doit être traitée comme un cas par défaut. Par exemple, la norme CGNS considère que l'absence d'un noeud de type GridLocation de la structure de données FlowSolution, noeud qui spécifie la localisation des solutions sur le maillage, doit être intérprétée comme un cas par défaut équivalent à «*vertex*». Nous n'acceptons pas cette approche, car nous considérons qu'il est une source d'erreurs et de confusion, en plus du fait qu'il rend plus difficile à lire et interpréter les informations contenues dans la base de données. Il faut penser (et l'histoire des standards le confirme - voir IGES) que, au delà d'une certaine limite, ce type de flexibilité masque des ambiguïtés. Cela conduit à des produits logiciels qui, paradoxalement, utilisent le même standard pour décrire et stocker les données, mais qui ne se «*comprennent* » pas [voir le Chapitre 2 sur IGES].

3.8 Le noeud ZoneBC

Le noeud ZoneBC contient les informations concernant les conditions aux limites pour une zone donnée. Les informations sont stockées séparément pour chaque

«*patch*» à l'aide de la structure de données BC, qui contient les informations concernant un «*patch*». Un «*patch*» BC représente le sous-domaine d'une face où les conditions aux limites sont appliquées. Si une zone contient N «*patch*», la structure ZoneBC aura une liste de N instances différentes de type BC. La structure BC contient le type de condition aux limites et les données qui définissent les conditions aux limites du «*patch*».

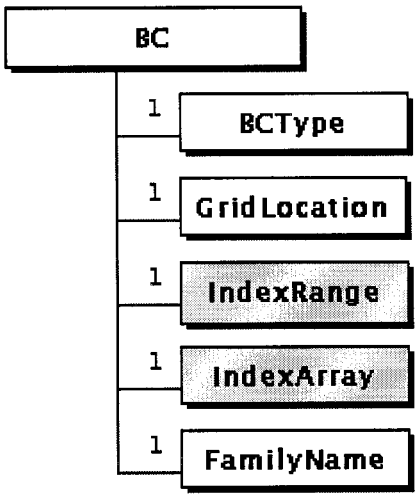


FIGURE 3.12 Structure de données qui définissent un patch

Les entités requises par Opale pour définir les «*patch*» sont :

BCType #
GridLocation #
PointRange ou PointList #
FamilyName #

Un sous-domaine est défini en utilisant les entités PointRange ou PointList. FamilyName spécifie la famille sur laquelle les conditions aux limites sont définies. Les conditions frontières peuvent être définies dans les familles. Dans ce cas, le type de condition aux limites, spécifié par BCType doit avoir la valeur «*FamilySpecified*»

[voir le SIDS, Section 12].

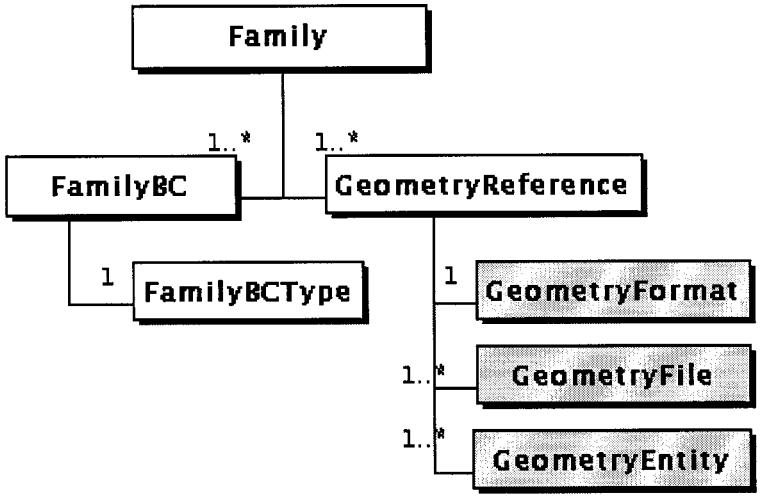


FIGURE 3.13 Structure de données Family requise par Opale

Le noeud ZoneBC	
Description	le noeud qui regroupe toutes les informations concernant les conditions limite relatives à une zone
Parent(s)	Zone
Enfants requis par Opale	- BC
Enfants optionnels	- ReferenceState # - DataClass # - DimensionalUnits # - Descriptor #
Données	aucune

3.9 Le noeud Family

Le SIDS définit la notion de famille comme un objet intermédiaire qui assure le découplage entre la géométrie et la base de données CGNS. Le maillage est lié

à une structure de ce type en attribuant le nom de la famille à un «*patch*» de condition limite (BC) ou à une zone. L'avantage le plus important de ce niveau intermédiaire est le fait que la densité du maillage et les entités géométriques peuvent être modifiées sans que l'on soit obligé de refaire l'association entre les noeuds de l'objet intermédiaire et la géométrie. La structure de données Family implémente ce concept. Nous avons imposé l'existence de cette structure au niveau de la base car nous avons besoin de connaître la géométrie. Le noeud GeometryReference est utilisé pour implanter le lien avec la géométrie en utilisant trois entités :

GeometryFormat #

GeometryFile #

GeometryEntity # [voir le SIDS page 136]

Le noeud FamilyBC contient un seul paramètre : BCType

Le noeud Family	
Description	Le noeud qui regroupe toutes les informations relatives à une famille CFD dans le sens défini par SIDS
Parent(s)	CGNSBase, Zone
Enfants requis par Opale	- GeometryReference # - FamilyBC #
Enfants optionnels	- Descriptor
Données	

3.10 Le noeud Descriptor

La structure de données Descriptor nous donne la possibilité de stocker des informations générales concernant la base de données. Nous considérons que des informations concernant la génération du fichier CGNS, l'auteur, le solveur utilisé et la version du fichier Opale, doivent être stockées dans la base de données. Dans le but de garder

toutes ces informations, un noeud de type Descriptor a été ajouté comme requis dans la base de données. Les informations qui seront gardées dans ce noeud sont :

l'auteur #

la date #

le fichier d'entrée +

le nom du solveur #

la version du fichier Opale #

Par définition, un noeud de type Descriptor ne peut contenir qu'une chaîne de caractères incluant les caractères spéciaux comme tab, nl, etc. Pour rendre claire et flexible la structure de données stockée ici, nous avons choisi de décrire les données en utilisant le format XML. En sérialisant le fichier XML contenant les informations dont nous avons besoin, nous obtenons une chaîne qui peut être stockée par un noeud Descriptor. Le noeud peut être généré de façon automatique lors de la phase d'optimisation du fichier CGNS. La structure proposé est la suivante :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<definitionsFile>
<auteur>
    <nom> LeNom </nom>
    <prenom> LePrenom </prenom>
</auteur>
    <date> jjmmaaaa </date>
    <fichierEntree> leFichier </fichierEntree>
    <solveur> leSolveur </solveur>
<versionFichierOpale> laVersion </versionFichierOpale>
</definitionsFile>
```

3.11 Le noeud ConvergenceHistory

Des informations concernant le nombre d'itérations du solveur et l'état de la convergence à chaque itération sont considérées très importantes pour le modèle Opale. CGNS contient le mécanisme nécessaire pour stocker ces informations. Conformément au SIDS, le noeud ConvergenceHistory a comme requis une seule entité : Iterations, qui stocke le nombre des itérations du solveur. Toutes les entités de type DataArray contenues par la structure de données ConvergenceHistory héritent de la valeur de ce paramètre, utilisé pour définir la longueur des tableaux de données. NormDefinition est un noeud de type Descriptor non-requis par la norme CGNS, mais fortement recommandé par le SIDS. Le format de données de ce noeud n'est pas standardisé ce qui nous a amené à définir le format de données comme une liste d'identificateurs qui donne le nom et l'ordre des paramètres qui se trouvent dans le noeud ConvergenceHistory.

Le noeud ConvergenceHistory	
Description	Noeud qui regroupe les informations concernant l'histoire de convergence
Parent(s)	CGNSBase, Zone
Enfants requis par Opale	- Iterations # - NormDefinitions #
Enfants optionnels	- DataClass - DimensionalUnits
Données	Les paramètres décrits dans le noeud Descriptor

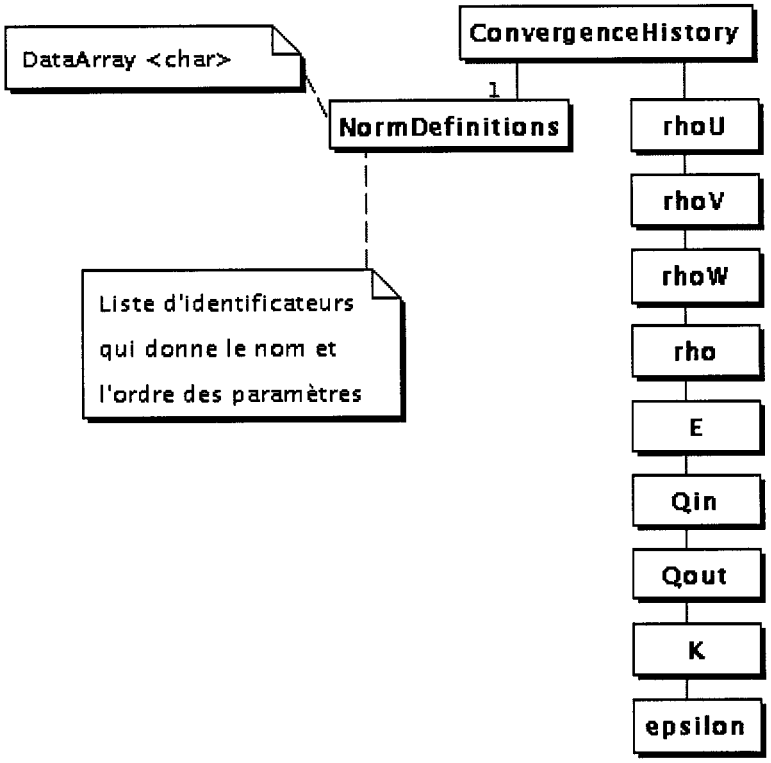


FIGURE 3.14 ConvergenceHistory

3.12 Le noeud SimulationType

Le SIDS prévoit la possibilité de stocker les informations concernant le type de simulation au niveau de la base et au niveau d'une zone sans donner d'information en ce qui concerne le choix qu'on doit faire. Les types de simulation qui peuvent être identifiés par cette structure sont : Null, UserDefined, TimeAccurate et NonTimeAccurate. Nous avons décidé d'imposer que l'information soit stockée toujours au niveau de la base, donc comme enfant du noeud CGNSBase.

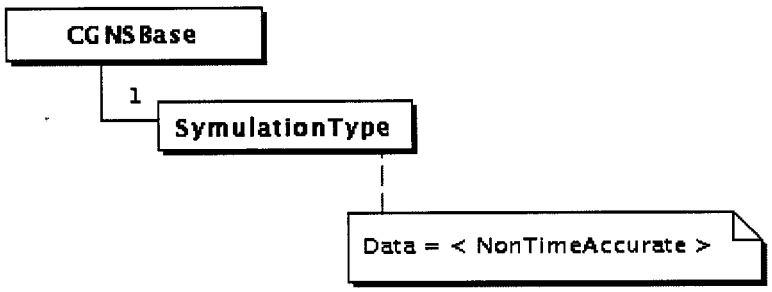


FIGURE 3.15 Structures de données définissant le type de simulation

3.13 Les paramètres de calcul

Parce que nous avons décidé de sauvegarder les paramètres de calcul utilisés dans le processus de simulation numérique (des paramètres spécifiques au solveur), nous avons introduit un noeud supplémentaire de type «*UserDefinedData*» qui stockera ces données. Les données sont stockées à l'aide des entités de type DataArray. Un exemple de données qui sont spécifiques au solveur NUMECA et qui sont stockées dans ce noeud sont :

AUTO_GRID_MESH
COARSE_FLAG
NUMBER_ OF_GRID_LEVEL
VERSION
NAME
FLUID_NAME
FLUID_TYPE
START_FROM_INITIAL_SOLUTION_FILE
BETPAR
CFLVIS
OMGSY
NSTAGE

NGRID

NBLADE

MTETUR

NFMGCY

LOCTST

KEGRID

Le noeud *ComputationParameters* est stocké comme enfant au niveau de la base.

3.14 Le noeud Gravity

L'entité Gravity est utilisée pour définir un vecteur gravitationel. Le seul enfant requis par la norme CGNS est la structure GravityVector qui contient les composantes du vecteur de la gravité.

Le noeud Gravity	
Description	Noeud qui regroupe les informations concernant le vecteur de gravité
Parent(s)	CGNSBase, Zone
Enfants requis par Opale	- GravityVector #
Enfants optionnels	- DataClass - DimensionalUnits
Données	Tableau ayant la dimension [1, PhysicalDimension] et qui définient le vecteur de gravité

3.15 Les noeuds non supportés par Opale

3.15.1 Le noeud DataConversion

DataConversion est une entité utilisée par la norme CGNS pour définir des facteurs de conversion dans le cas où les données stockées dans une entité de type DataArray sont non dimensionnelles. Opale ne supporte pas les entités de ce type car nous considérons que des données non-dimensionnelles sont une source importante d'erreurs. Donc, pour pouvoir être sauvegardées dans un fichier CGNS supporté par Opale, les données doivent subir un prétraitement.
Par ailleurs, les noeuds suivants ne sont pas pertinents pour le projet, et ne sont pas supportés par le modèle de données :

- GasModel
- ThermalConductivityModel
- BaseIterativeData
- RigidGridMotion
- ArbitraryGridMotion

CHAPITRE 4

IMPLANTATION ET UTILISATION DU MODÈLE THÉORIQUE

4.1 Systéme de représentation de données utilisant une couche de grammaire

En évaluant les approches existantes pour représenter des informations numériques, du point de vue de l'analyse numérique, nous avons pu constater que toutes les normes sont structurées de la même manière. De plus, toutes les normes essayent de définir des formats neutres (donc, avec un minimum des contraintes) en laissant les applications implanter les contraintes spécifiques à chaque domaine particulier.

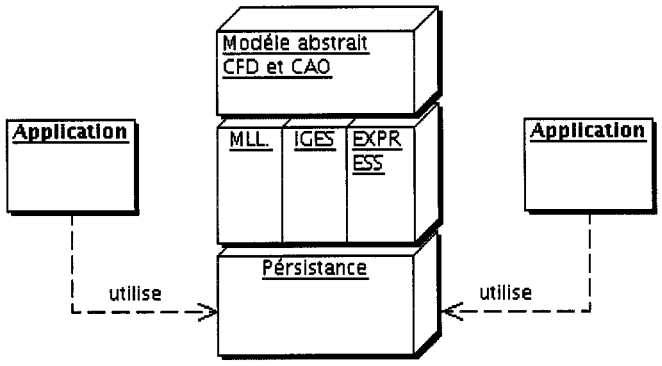


FIGURE 4.1 Les couches composant un système de représentation des données CFD et CAO

Le problème de base avec les standards basés sur des formats neutres, comme CGNS, STEP et IGES, est la duplication des données et l'ambiguïté de l'information due au modèle de données très général et à l'impossibilité d'imposer des contraintes spécifiques à un domaine d'application donné.

Nous avons identifié l'existence des trois couches principales que les normes actuelles définissent [voir fig 4.1] :

- la couche abstraite
- la couche interface
- la couche pérsistance

La couche abstraite est la couche de plus haut niveau, qui définit le contenu intellectuel des données CFD ou CAO et l'organisation de la structure de données. Étant donnée le fait que le SIDS définit actuellement le plus complètement la structure d'une base de données CFD, nous avons décidé de l'utiliser comme référence. De plus en plus, les modèles de données CFD développés et la façon de les représenter physiquement sont basées sur le SIDS, ce qui est un indice de plus de sa maturité. CGNS est d'ailleurs en voie d'être integré à la norme STEP.

La couche intermédiaire est la couche qui permet à un modèle purement conceptuel de se matérialiser. C'est la couche qui assure le passage du concept à l'implémentation. C'est à ce niveau que les contraintes structurales et comportamentales sont implémentées.

La couche pérsistance est la couche que les outils et les applications, qui théoriquement font partie d'un système d'analyse numérique, utilisent pour partager et échanger des données. La couche pérsistance définit la façon concrète dont les données sont stockées, le format de fichiers.

La couche abstraite définit un ensemble de caractéristiques et de règles de comportement généralement valables pour les domaines de la CFD et de la CAO. Mais, chaque domaine d'application a des caractéristiques biens spécifiques. La tâche de distinguer entre les caractéristiques particulières de chaque domaine est laissée à la couche suivante, c'set-à-dire la couche intermédiaire. En règle générale, la couche interface implémente le modèle abstrait au complet, exception faite de STEP qui utilise EX-

PRESS pour définir des comportements spécifiques. Donc elle hérite du modèle son comportement général. En particulier elle est incapable de définir les caractéristiques particulières à un domaine d'application donné. Les standards, transférent la responsabilité de gérer les données et la façon de les interpréter, aux utilisateurs. D'où l'incompatibilité entre des applications qui utilisent le même format de description de données.

Nous avons conclu de l'analyse des approches existantes que l'architecture est incomplète et elle doit intégrer une couche supplémentaire au niveau de la couche intermédiaire, la couche grammaire. Celle-ci doit pouvoir définir des sous-ensembles de contraintes spécifiques aux domaines d'applications particuliers. De plus, la couche doit être facilement configurable. En passant par cette couche intermédiaire, qui définit des contraintes et les comportements spécifiques, nous sommes assurés que la structure de données est claire et bien définie. De plus, avec une grammaire, nous sommes sûr qu'il n'y aura pas de données corrompues lors d'échange de données si chaque participant connaît le dictionnaire de données, donc la structure à laquelle les données doivent absolument se conformer.

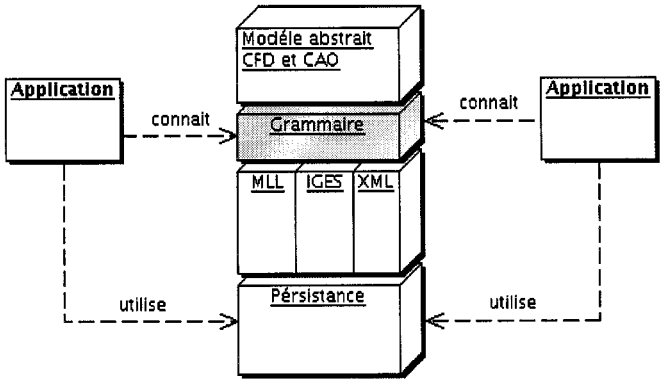


FIGURE 4.2 L'architecture en quatre couches proposée et la façon dont les applications interagissent

La figure 4.2 montre l'approche qui devrait être employé pour implanter un standard qui assure un échange de données propre et sans pertes.

Dans le cas standard montré dans la figure 4.1, les applications n'ont aucune connais-

sance concernant les éventuelles contraintes spécifiques à un domaine d'application donné. Tous les fichiers de données sont traités de la même manière. De plus, il n'y a aucun façon de vérifier si les données ne sont pas altérées.
Dans l'approche que nous proposons, les applications connaissent la couche grammaire, ce qui leur donne la possibilité de connaître à priori la structure qu'elles utilisent (sortie ou entrée) et de la valider. De plus, en définissant la couche grammaire pour un domaine d'application donné, nous pouvons générer des données parfaitement compatibles dans des formats de représentation de données différents.

La norme CGNS utilise comme niveau intermédiaire la bibliothèque MLL. Cette bibliothèque gère et implémente très bien les contraintes définies par le SIDS. Mais, le SIDS définit des règles assez générales qui ne peuvent pas imposer des contraintes spécifiques à chaque domaine d'application. Si on définit un sous-ensemble de règles de comportement et structurales spécifiques à un domaine donné (par exemple les turbines hydrauliques), il faut modifier le MLL pour l'implémenter. De plus, toutes les applications qui utilisent cette structure devraient utiliser le même MLL. Ce qui serait impraticable.

L'implantation effective de la couche grammaire que nous avons proposée peut être réalisée à l'aide de XML, qui s'y prête très bien à cause des dictionnaires et grammaires DTD et XSD qui lui sont spécifiques. Dans cadre du projet, nous avons choisi d'utiliser l'implémentation avec un DTD parce que, dès le début du projet, il y avait déjà un modèle de données DTD basé sur le SIDS. Le dictionnaire existant a été modifié en fonction des contraintes spécifiques au domaine des turbines hydrauliques. Pour plus de souplesse, le dictionnaire est composé de trois DTD [figure 4.3] : un pour définir les contraintes structurales ; un pour définir le nom d'attributs qui définissent le nom des éléments CFD ; et un pour définir le nom d'attributs qui définissent les données contenue dans le noeud lui-même.

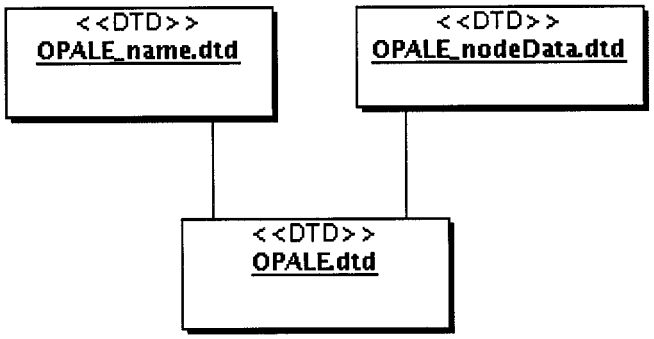


FIGURE 4.3 La stucture du DTD

La nécessité d'avoir une couche intermédiaire entre le niveau conceptuel abstrait et la couche qui gère l'interaction avec la base de données dédiées à l'analyse numérique est évidente. Le domaine de l'analyse numérique est très vaste. Ce que d'habitude on appelle modèle de données dans le domaine de l'analyse numérique est plutôt un meta-modèle de données car il englobe des données assez générales provenant de divers domaines d'application. Pour que des applications puissent communiquer, elles doivent disposer d'une infrastructure bien définie. Pour que cette infrastructure soit utilisable, il faut imposer des contraintes tant au niveau structural qu'au niveau comportemental, ce qui n'est pas possible avec un passage direct du meta-modèle à l'implantation.

L'idéal serait que les applications (solveur, analyseur, optimiseur, etc) puissent utiliser un dictionnaire de donnée (une grammaire) dans le but d'accomplir la tâche de sauver ou de lire des structures de données.

4.2 Processus de standardisation d'un fichier CGNS

Présentement, plusieurs résoluteurs, parmi lesquels on peut mentionner CFX et NUMECA, ont la capacité de sauvegarder les résultats d'une simulation numérique dans le format CGNS. Il y a aussi des traducteurs *«maison»* qui traduisent en CGNS un fichier de données issues d'une simulation numérique avec des résoluteurs comme

TaskFlow.

Bien que conceptuellement nous puissions dire que les données stockées dans des fichiers CGNS générés par différents logiciels commerciaux ou *«maison»* sont "compatibles", il y a encore de très grandes différences au niveau de la structure et du contenu de ces données selon l'application qui les a générées.

Un certain nombre de variables ne sont pas pertinentes pour des simulations d'écoulement de fluides dans le contexte des turbines hydrauliques. La densité, la température, la viscosité dynamique sont des parametres constants. Or, ils apparaissent tous sous la forme d'un scalaire à chaque noeud (une variable), ce qui ne fait que grossir inutilement le fichier CGNS.

D'autres informations pertinentes du point de vue des simulations d'écoulement dans des turbines comme la gravité, l'état de référence, l'histoire de convergence, les paramétrés de calcul, etc. manquent au fichier CGNS.

Étant donné que les fichiers générés par les différents résoluteurs sont plus ou moins complets et conformes aux spécifications de la norme CGNS, nous avons décidé de diviser le problème en trois étapes :

1. le niveau conceptuel,
2. le niveau traduction/adaptation et
3. le niveau de calcul.

La première étape correspondant à la couche abstraite d'OPALE, vise à définir la structure d'un fichier *«standard»*, dans le contexte de d'analyse des turbines hydrauliques [voir chapitre 3]. Sélectionner toutes les informations qui nous sont nécessaires pour accomplir n'importe quelle simulation de type rotor/stator et structurer ces données en respectant les spécifications du SIDS (SIDS - Standard Interface Data Structure).

La deuxième étape consiste à transférer (générer) de façon automatique les fichiers CGNS provenant de l'un ou l'autre des résoluteurs dans la structure standard définie

à la première étape. L'utilisation d'une grammaire pour définir le comportement et la structure des données était évidente.
Le fait qu'on ait séparé le modèle de l'implémentation nous a donné beaucoup plus de flexibilité. De plus, l'utilisation du langage XML pour implanter le modèle permettra eventuellement d'intégrer des changements au niveau conceptuel, de façon aisée.
L'approche que nous avons développée, consistant à définir la structure d'une base de données avec des mécanismes spécifiques à XML, nous donne un autre avantage quand on veut vérifier si une base de données quelqune est conforme ou non à un modèle prescrit. Pour faire la vérification et repérer les non conformances, il suffit de récupérer la structure de la base de données (*adf2xml*) et de lui appliquer le dictionnaire du modèle.

Dans la troisième étape, le fichier CGNS standard est utilisé par une application pour calculer le rendement hydraulique par bilan énergétique.
Les avantages issus du fait qu'on a utilisé XML pour définir la structure de données se sont rapidement faits sentir. En utilisant le même dictionnaire de données, nous avons bâti des structures de données XML pouvant être validées et capable de guider le processus de calcul.

Dans ce qui suit, on détaillera du point de vue conceptuel la deuxième étape, celle de la standardisation de la base de données CGNS, en décrivant l'architecture du processus de traitement des données.
Il faut souligner le fait qu'une des caractéristiques importantes du processus de standardisation d'une base de données CGNS est qu'il doit être à la fois indépendant de la structure originale de la base de données et automatique.
Il faut mentionner le fait que même si la base de données CGNS est basée sur une structure arborescente, assurée par l'*ADF(Advanced Data Format)*, les API dont elle dispose pour accéder et manipuler les données qui y sont stockées ne donnent

pas la possibilité de voir et de traiter la structure de données comme une structure arborescente. Ceci a pour résultat qu'il n'y a aucun façon de parcourir itérativement la structure arborescente. Ces contraintes, liées à la façon dont les données sont stockées et le faible couplage imposée par les caractéristiques d'un tel processus, nous ont amené à un processus complexe.

Étant donné le fait que la standardisation d'une base de données CGNS est un processus complexe, il peut être décomposé en plusieurs sous-processus.

Les sous-processus constituant les étapes de mise en forme de la base de données que nous avons identifiés sont :

1. Générer le fichier XML correspondant à la structure de données du fichier CGNS original (adf2xml).
2. Appliquer le modèle de données OPALE sur la structure de données obtenue.
3. Utiliser la structure obtenue pour guider le processus de mise en forme standard du fichier CGNS.

Nous avons choisi d'utiliser XML comme gestionnaire de la base de données (dans les processus de validation, de filtrage, d'ajout de données et de mise en forme), à cause de sa généricité et de sa flexibilité en ce qui concerne la modélisation et la description des structures de données. Si nous voulons ajouter, enlever ou modifier des données il sera suffisant de modifier le fichier de données XML correspondant pour que la structure du fichier cible, CGNS, soit modifiée. Pour vérifier que les fichiers de données sont grammaticalement corrects, nous utiliserons un langage de définition de grammaire XML. Aujourd'hui, il en existe deux. Le premier est le langage DTD (Document Type Definition) : historiquement, ce fut le premier proposé. Le dèuxieme, initialement développé par la société Microsoft, est le langage de schémas (il s'agit en fait d'un langage XML dédié à la définition de grammaire). Nous allons, dans ce chapitre, nous concentrer uniquement sur la définition et l'utilisation d'un DTD.

4.2.1 Extraction de la structure du fichier CGNS (ADF2XML)

La première étape, adf2xml, consiste à générer le fichier XML correspondant à la structure de données du fichier CGNS. La figure 4.4 montre le processus d'extraction en format XML de la structure d'une base de données CGNS.

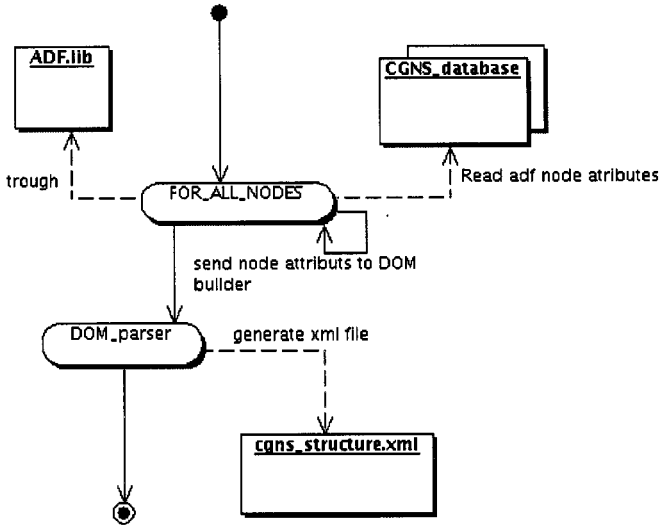


FIGURE 4.4 Génération de la structure du fichier CGNS en XML

La figure 4.5 montre la structure d'un fichier CGNS produit par NUMECA et visualisé à l'aide d'outil *adfviewer.*

En sachant que la base de données CGNS est bâtie en utilisant une structure d'arbre binaire, nous avons choisi d'utiliser un arbre binaire bâti à l'aide de l'analyseur DOM de XML pour exprimer la structure de la base de données CGNS. La structure de la base de données CGNS est lue en passant par l'ADF (Advanced Data Format) car c'est la seule façon de traiter le fichier CGNS comme un arbre. Au fur au mesure qu'un noeud CGNS est lu, il est passé à une *usine* d'entités XML de type *DOMElement.* Les entités XML construites sont ajoutés par la suite dans l'arbre DOM correspondant. L'arbre DOM est bâti à partir de zéro, en se basant sur un dictionnaire de données (DTD). Une fois la structure de la base de données CGNS

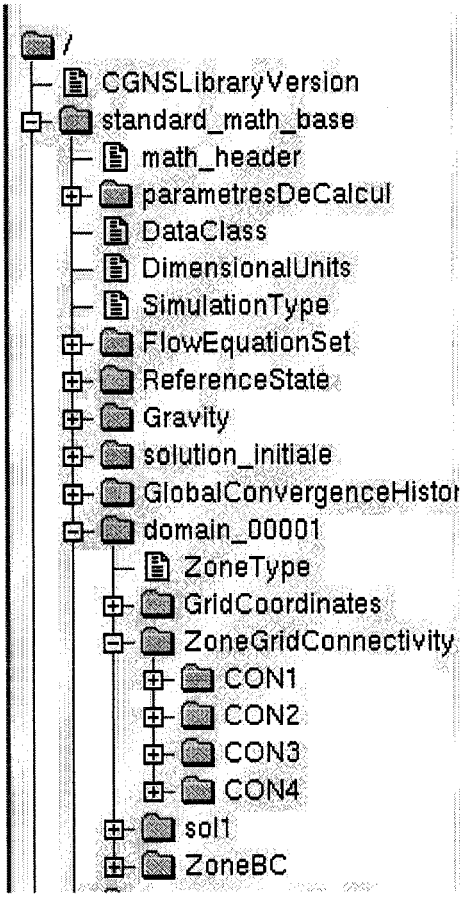


FIGURE 4.5 La structure du fichier CGNS

lue, en utilisant les mécanismes d'analyseur DOM, on génère le fichier XML correspondant. Pour chaque type de noeuds, la norme CGNS définit des étiquettes et des noms bien spécifiques. Nous avons choisi que les noms des balises XML soient le nom de l'étiquettes CGNS et, pour bien spécifier chaque entité, son nom sera utilisé comme attribut XML. Car il y a des noeuds parents CGNS qui contiennent eux-mêmes des données que nous avons décidés d'ajouter comme des attributs dans les balises. La balise XML aura donc la structure suivante :

< NomEtiquette_t name= "nomNoeud" nodeData= "data" >

Une autre information qui on pourrait vouloir chercher ou ajouter comme attribut à une balise XML est le type de données stockées dans un noeud. Ce type d'information

peut être très pertinent dans le cas où le dictionnaire est défini avec des schémas car il peut être utilisé pour une validation supplémentaire.

La figure 4.6 montre la structure XML d'un fichier CGNS produit par le logiciel adf2xml.

```
<?xml version="1.0" encoding="UTF-8" standalone=
<OPALE.metadata>

  <CGNSLibraryVersion_t name="CGNSLibraryVersion
EO1</CGNSLibraryVersion_t>

  <CGNSBase_t name="standard_math_base">
    <Descriptor_t name="math_header"/>
    <DataClass_t name="DataClass"/>
    <DimensionalUnits_t name="DimensionalUnits"/
    <SimulationType_t name="SimulationType"/>
    <UserDefinedData_t name="solution_initiale">
      <Descriptor_t name="Commentaire"/>
    </UserDefinedData_t>
    <Zone_t name="domain_00001">
      <ZoneType_t name="ZoneType"/>
      <GridCoordinates_t name="GridCoordinates">
        <DataArray_t name="CoordinateX"/>
        <DataArray_t name="CoordinateY"/>
        <DataArray_t name="CoordinateZ"/>
      </GridCoordinates_t>
      <ZoneGridConnectivity_t name="ZoneGridConn
        <GridConnectivity1to1_t name="CON1">
          <Transform name="Transform"/>
          <IndexRange_t name="PointRange"/>
          <IndexRange_t name="PointRangeDonor"/>
        </GridConnectivity1to1_t>
        <GridConnectivity1to1_t name="CON2">
          <Transform name="Transform"/>
          <IndexRange_t name="PointRange"/>
          <IndexRange_t name="PointRangeDonor"/>
        </GridConnectivity1to1_t>
        <GridConnectivity1to1_t name="CON3">
          <Transform name="Transform"/>
```

FIGURE 4.6 La structure du fichier CGNS en XML

Il est possible de lire et de transcrire en XML encore plus d'informations sur chaque noeud. Le processus de validation revient à passer à l'analyseur DOM un dictionnaire de données (DTD ou SCHEMA). La sortie de ce processus est un fichier XML qui

contient la structure de données réelle telle qu'elle est instanciée dans le fichier CGNS.

4.2.2 L'analyseur de structure

L'étape suivante consiste à analyser syntaxique la structure d'une base de données CGNS. L'analyse est faite en utilisant l'image XML de la structure de la base de données CGNS, un fichier contenant des informations specifiques à un résoluteur donné et le dictionnaire de données (DTD) d'OPALE [voir la figure 4.7].

Les contraintes qui ont influencé notre choix sont les suivantes :

- Le fichier de données CGNS est un fichier binaire, donc pour y accéder, il faut passer par un gestionnaire de base de données,
- La structure de la base de données telle qu'elle est définie dans le SIDS n'est pas très restrictive, ce qui génère beaucoup de cas possibles,
- La structure de données OPALE, même si elle est basée sur la norme CGNS, est beaucoup plus restrictive et, de plus, la norme CGNS telle qu'elle est aujourd'hui n'est pas encore dans sa forme finale, ce qui générera certainement des changements à ce niveau.

Trois sortes d'opérations peuvent être définies sur une base de données CGNS :

1. Ajout d'information <OPALE.add>
2. Retrait d'information <OPALE.delete>
3. Déplacement d'information <OPALE.move>

Si des informations pertinentes pour nos applications d'analyse numérique manquent à la base de données, l'analyseur ajoutera au fichier XML une balise de commande *<OPALE.add>* suivi par la balise qui décrit le noeud en cause (qui est bâti en utilisant le même DTD), exactement à l'endroit où le noeud en cause manque.

L'information à ajouter peut être lue à partir d'un autre fichier XML et insé-

rée directement dans le fichier XML optimisé, entre les balises *<OPALE.add>* et *</OPALE.add>*, ou la balise peut indiquer une référence vers le fichier XML qui contient l'information dont on a besoin. Par exemple, les informations concernant la description des équations utilisées (FlowEquationSet_t) pour une simulation numérique sont absentes dans toutes les bases de données CGNS générées par les resoluteurs commerciaux. Nous avons créé un fichier XML nommé flowEquationSet.xml (*«solver specific»* dans la figure 4.7), qui décrit les données manquantes de la base de données comme : «GouverningEquations», «ViscosityModel» et «TurbulenceClosure». Un autre fichier XML, referenceState.xml, contient des informations concernant la température, le nombre de Reynolds, la densité, etc. qui sont stockés comme des scalaires dans chaque noeud au niveau du «FlowSolution», alors que dans la structure conforme au standard, elles sont stockées au niveau du noeud ReferenceState. Si un noeud ou un ensemble de noeuds doit être enlevé du fichier CGNS, on utilise une balise *«OPALE.delete»* pour indiquer les noeuds à enlever. Le résultat

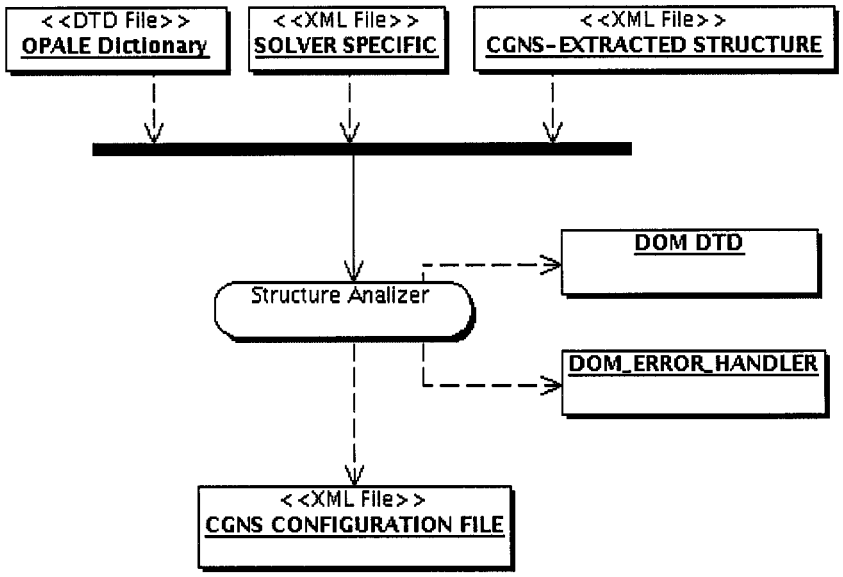


FIGURE 4.7 Génération du fichier de configuration

de cette étape est un fichier XML de configuration qui contient des opérations qui doivent être appliquées sur la base de données CGNS.

Le logiciel *CGNS Analyzer*, que nous avons développé, permet d'ajouter ou enlever des noeuds à travers d'une interface graphique. Il est possible d'ajouter un noeud à la fois en spécifiant son nom, l'étiquette et la valeur, où on peut ajouter des structures des données qui sont stockées dans des fichiers XML.

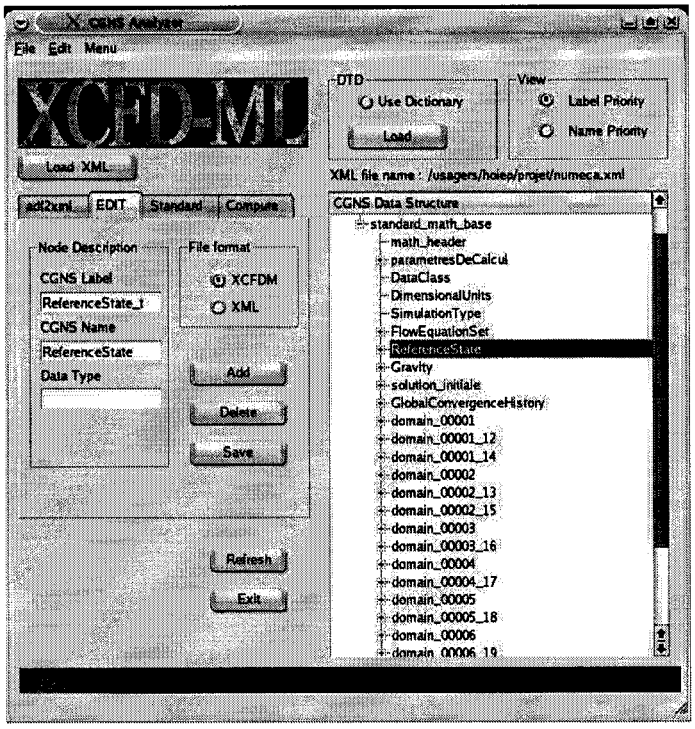


FIGURE 4.8 L'interface graphique qui gère le processus de création du fichier XML de configuration

La figure 4.8 montre l'interface qui gère le processus de création du fichier XML de configuration. La balise de commande *<OPALE.add>* qui spécifie qu'un nouveau noeud ou une structure doit être ajoutée dans le fichier CGNS est incluse par défaut dans le fichier XML.

Si un noeud de type ReferenceState doit être ajouté dans la base de données CGNS,

le fichier XML de configuration de la base de données aura la structure suivante :

.....

<OPALE.add>

<ReferenceState_t name="ReferenceState" >

<Descriptor_t name="ReferenceStateDescription">Export mesh and solution data from NUMECA<Descriptor_t>

<DataArray_t name="Reynolds" type="R8" >2.66e+006</DataArray_t>

<DataArray_t name="Temperature" type="I4" > 288 </DataArray_t>

.....

</ReferenceState_t >

</OPALE.add> .....

4.2.3 Mise en forme du fichier CGNS

Dans ce qui suit, on détaillera la troisième étape, celle de mise en forme effective du fichier CGNS.

Une fois le fichier XML filtré et enrichi avec les données pertinentes à nos applications, et en se basant sur un dictionnaire de données (DTD ou SCHEMA) décrivant la structure standard d'une base de données CGNS, la tâche de restructuration du fichier CGNS peut être lancée.

Le standardiseur reçoit comme entrée le fichier XML de configuration qui conduira le processus, un fichier auxiliaire propre à un résoluteur donné et la base de données CGNS à standardiser. La figure 4.9 montre le processus de mise en forme d'une base de données CGNS.

Bien que l'utilisation de l'approche DOM parâit plus appropriée à cause de la structure arborescente de la base de données CGNS, nous avons décidé d'utiliser l'approche séquentielle de SAX pour les raisons suivantes :

- l'impossibilité de définir de façon unitaire des opérations de manipulation des données sur les entités CGNS ;
- l'accès à une entité donnée de l'arbre est faite de façon séquentielle.

Le fichier de configuration XML sera analysé de façon séquentielle, tout en mettant à jour la base de données CGNS en concordance avec les opérations décrites dans ce fichier de configuration. En analysant le fichier XML de standardisation, et en

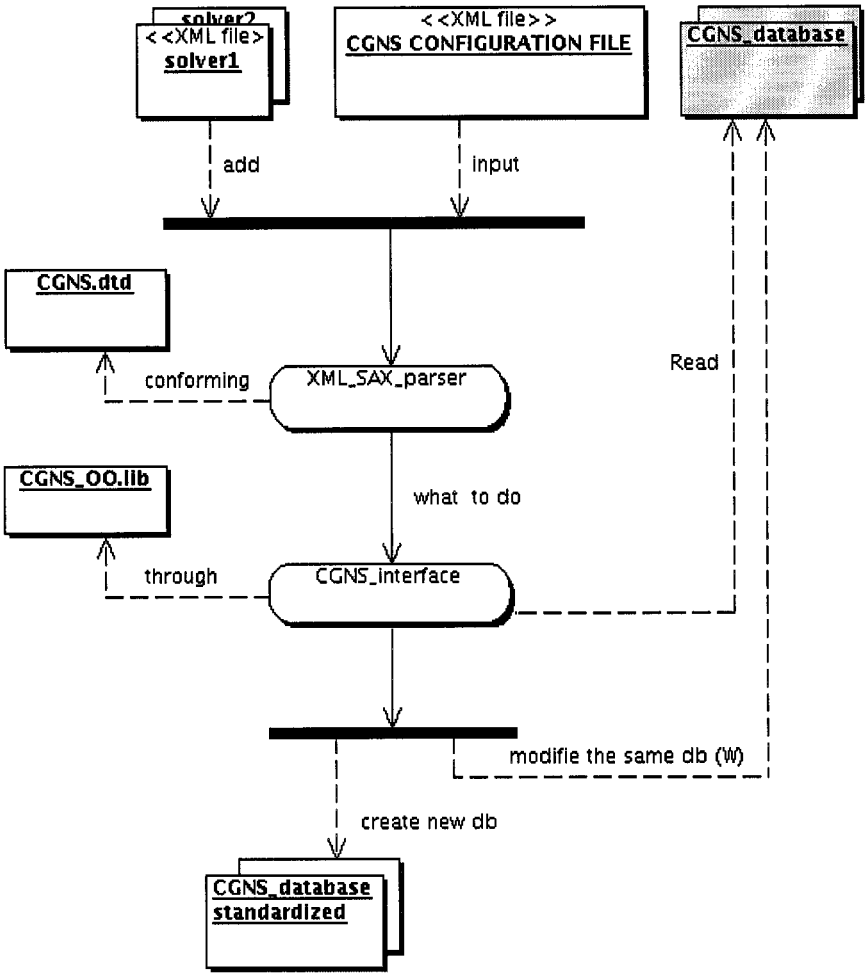


FIGURE 4.9 Le processus de standardisation

interprétant les balises de commande trouvées, le logiciel configure la base de données CGNS conformément au modèle OPALE.

Le logiciel *opale2standard* a une structure basée sur l'analyseur SAX dont il hérite son comportement. De plus, il implémente la structure et le comportement des noeuds CGNS.

Le logiciel est bâti autour de la classe abstraite *SAX2CgnsHandler* qui hérite de la classe *DefaultHandler* de SAX et d'un gestionnaire des comportement, *XMLHandlerManager* [voir la figure 4.10]. Nous avons choisi d'utiliser cette structure, car chaque entité CGNS a des caractéristiques particulières en ce qui concerne son comportement, sa structure, son façon d'y accéder, les opérations d'écriture et de lecture. La classe de base *SAX2CgnsHandler* définit un comportement général valable pour toutes les entités CGNS, en laissant les particularités être définies au niveau de chaque classe dérivée. Donc, à chaque entité CGNS correspond une classe qui hérite de la classe de base *SAX2CgnsHandler*, et qui implémente ses particularités. L'analyse du fichier XML, est initiée par l'intermédiaire d'une classe *SAX2CgnsMainHandler* qui gère aussi les fichiers XML à traiter.

L'analyseur SAX, étant basé sur des évènements, laisse la gestion de ces évènements à l'application qui l'utilise. La gestion des «handlers» est implantée à l'aide d'une pile par la classe *XMLHandlerManager*. Toutefois, la classe *XMLHandlerManager* gère l'instanciation des objets de type *SAX2CgnsHandler* par l'intermédiaire d'une usine d'objets.

Lorsqu'une balise XML est rencontrée par l'analyseur SAX, un objet de type approprié est généré et ajouté dans une liste. La classe *SAX2CgnsHandler* contient une liste d'objets de type *SAX2CgnsHandler* qui stocke des pointeurs vers les objets décrivant les enfants de cette entité. Chaque objet de ce type contient des références vers le noeud approprié de la base de données CGNS. Donc, si on veut accéder aux données stockées par un noeud ou un ensemble de noeuds, il suffit de chercher la référence voulue dans la structure créée.

En utilisant cette structure, la gestion des balises de commande est assurée par le

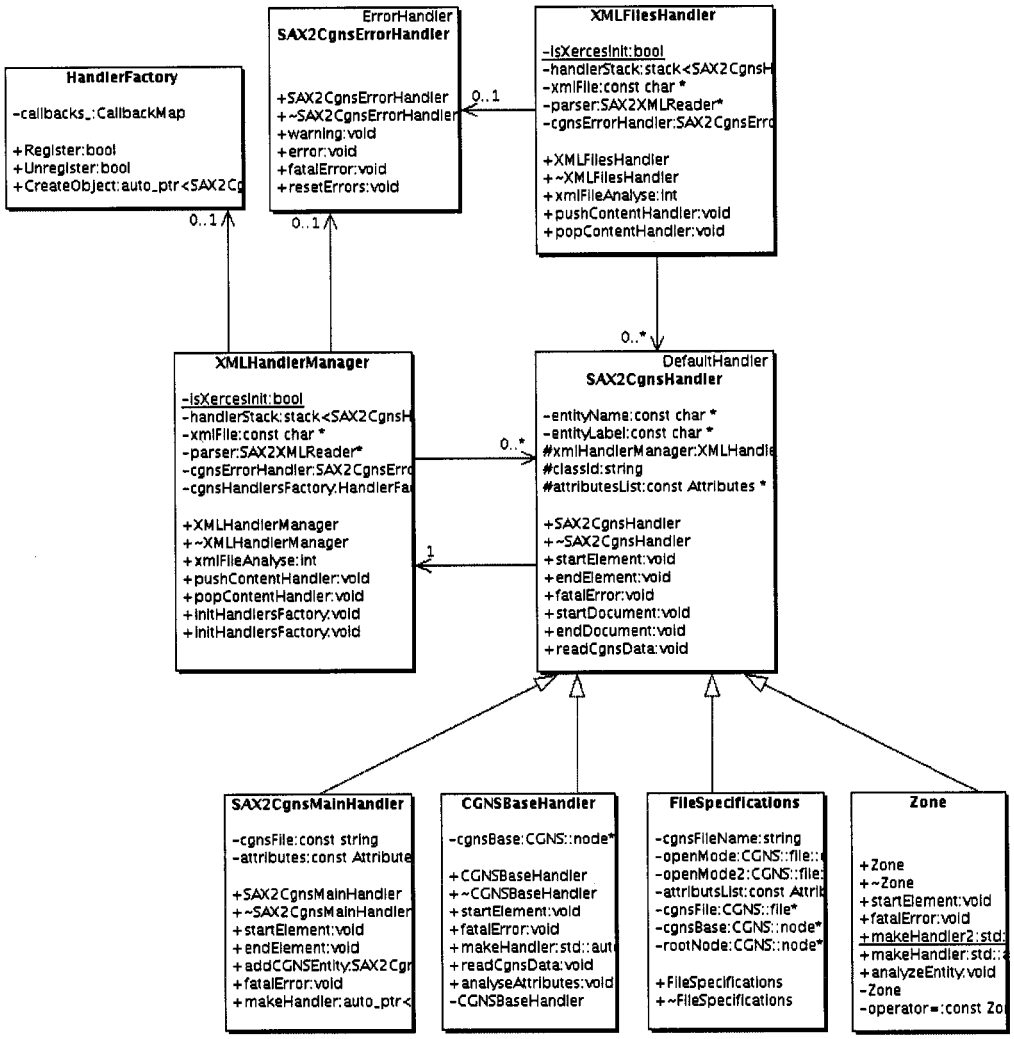


FIGURE 4.10 La diagramme des classe simplifiée du logiciel

même gestionnaire. Lorsqu'une balise de commande est rencontrée par l'analyseur, le gestionnaire change la façon de réagir du logiciel, en le mettant en mode commande.

CHAPITRE 5

CALCUL DU RENDEMENT HYDRAULIQUE EN UTILISANT UNE BASE DE DONNÉES CGNS

Le but de ce chapitre est de valider le modèle de données basé sur CGNS, à travers l'implantation d'une application réelle permettent de calculer le rendement d'une turbine hydraulique.

5.1 Considérations mathématiques

Dans ce qui suit, nous détaillons la méthode adoptée pour évaluer le rendement d'une turbine hydraulique. Il y a plusieurs façons d'évaluer le rendement, parmi lesquelles on retrouvent le calcul de ΔP (bilan énergétique entre l'entrée et la sortie) et le calcul de perte volumique (intégrer le tenseur de contrainte sur tout le volume). La méthode retenue est celle basée sur le bilan énergétique.

Le rendement de la turbine hydraulique peut être défini ainsi :

$$\eta = \frac{E_{roue}}{g\Delta H_{turb}} = \frac{P_{roue}/(\rho Q)}{g\Delta H_{turb}} \tag{5.1}$$

La chute sur les surfaces d'entrées ou de sortie est calculée par :

$$gH_A = \frac{\int_A (\frac{P}{\rho} + \frac{\vec{u}\cdot\vec{u}}{2})\rho\vec{u}\cdot\hat{n}\,dA}{\int_A \rho\vec{u}\cdot\hat{n}\,dA} \tag{5.2}$$

Le couple sur la roue est calculé en intégrant le couple effectif sur la surface des

aubes :

$$P_{roue} = \vec{\tau} \cdot \vec{\omega} = -N_P \omega \int_A (\vec{r} \times P\hat{n})_Z \, dA \tag{5.3}$$

Où on a :

- H = la hauteur
- gΔH = la quantité qui représente la variation de la chute nette

$$g\Delta H = gH_{entree} - gH_{sortie} \tag{5.4}$$

- Q = débit volumique
- ρQ = débit massique
- ρ = Densité de l'eau
- P_{roue} = Puissance générée par la roue
- P = Pression
- u = Vitesse de l'eau en repère fixe
- n = Normale unitaire à la surface
- ω = Vitesse de rotation de la roue (s^{-1})
- τ = Couple hydraulique sur la roue
- N_P = Nombre de passages dans la roue

5.1.1 Les hypothèses

Le calcul de pertes à été fait a partir d'une simulation d'une turbine de type *Francis* [voir 5.1], faite à l'IREQ, avec le résoluteur NUMECA.
La figure 5.2 montre, de façon schématique, une pale de la turbine et une directrice, tandis que la figure 5.3 montre la géométrie réelle de la pale avec la face d'entrée et la face de sortie.

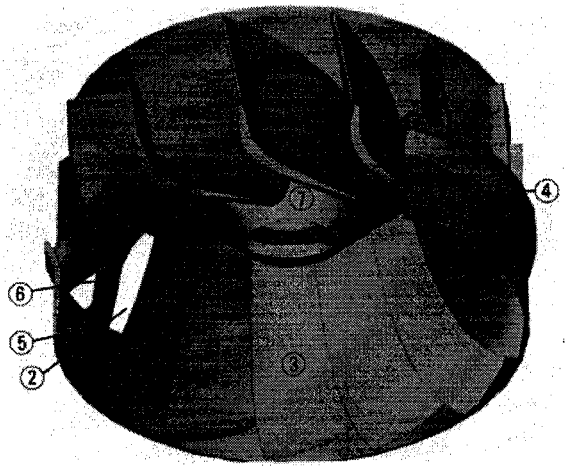


FIGURE 5.1 Turbine de type *francis*

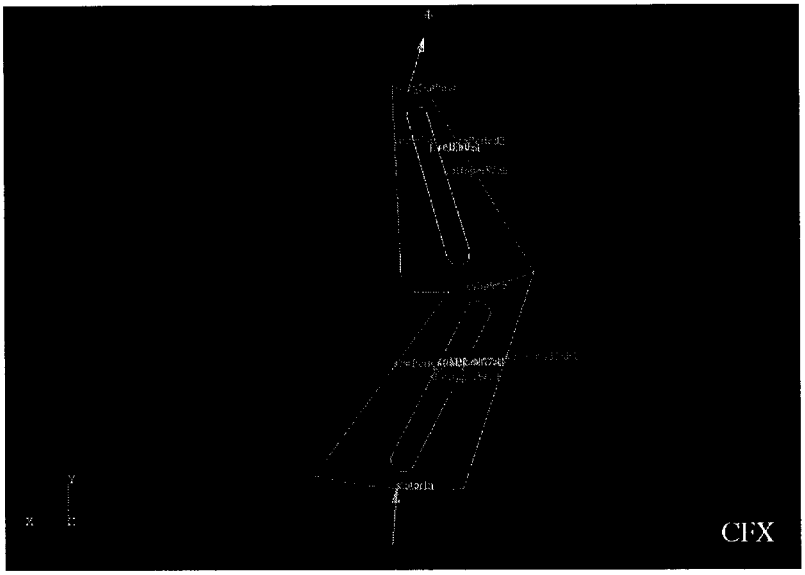


FIGURE 5.2 Une pale et une directrice d'une turbine

Nous avons posé les hypothèses suivantes dans le processus de calcul des pertes :

- L'axe de la turbine coïncide avec l'axe Z.
- La surface d'entrée pour calculer le bilan doit être située au même endroit que la surface de sortie de la roue (elle ne coïncide pas toujours avec l'entrée du domaine de calcul).
- Les intégrales doivent être calculées en interpolant sur les éléments de surface for-

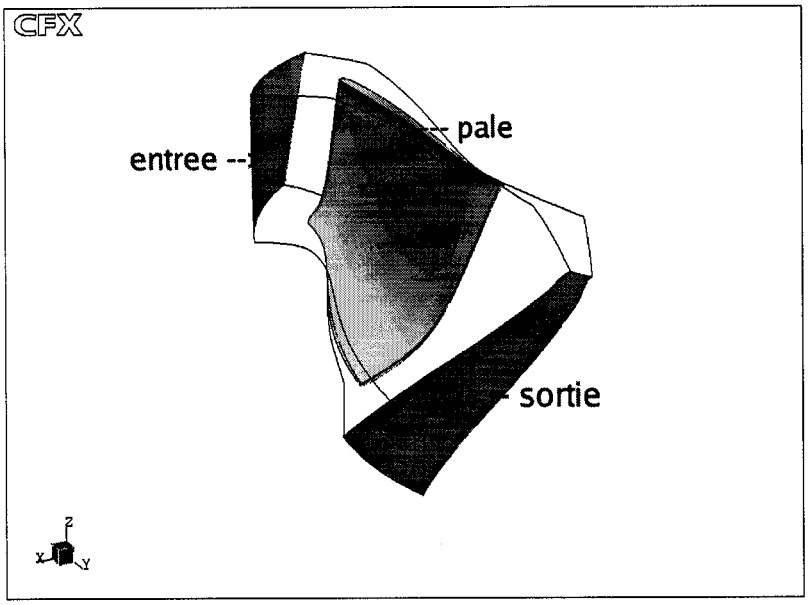


FIGURE 5.3 Pale de turbine *francis* en NUMECA

mant la surface d'intégration. Dans le cas d'une solution aux noeuds du maillage, on utilise une interpolation linéaire.

- Le calcul dans la roue est fait dans le repère tournant. Il est possible que la solution présente dans le fichier CGNS ne contienne pas les vitesses dans le repère absolu.
- Les normales utilisées sont des moyennes calculées à partir du maillage surfacique.
- Les régions d'entrée, sortie et la surface de l'aube doivent pouvoir être retracées à partir des données présentes dans le fichier CGNS. Elles peuvent couvrir plus d'une zone et peuvent représenter seulement une portion de la surface d'un bloc structuré.

L'analyse de cette application nous a mené à découper le problème en plusieurs sous-étapes :

- Définir les équations mathématiques dont on a besoin pour calculer le rendement
- Identifier et repérer l'information dont on a besoin dans la base de données CGNS
- Trouver la façon de repérer les faces géométriques qui composent une surface donnée dans la base de données CGNS
- Calculer le rendement par bilan énergétique.

5.2 Identifier et repérer l'information

La première étape dans le processus de calcul du rendement a consisté à repérer, dans le fichier de solutions CGNS, l'information dont nous avons besoin. Il fallait vérifier la façon dont elle est stockée, si elle est explicite ou s'il faut la calculer à partir d'autres informations présentes. Un autre aspect qui faisait partie de cette phase d'analyse à été d'identifier comment nous pouvons extraire toutes les valeurs requises à l'évaluation à partir d'une base de données CGNS.

5.3 Identifier les faces géométriques qui compose une surface

Pour valider le modèle, on évaluent le rendement par bilan énergétique, en calculant la perte d'énergie. L'énergie est calculée à partir d'un fichier de solutions CGNS. Le calcul de rendement peut être fait entre l'entrée et la sortie du domaine, mais il est possible de choisir de calculer le rendement entre deux surfaces qui ne sont pas nécessairement l'entrée et la sortie du domaine.
Une base de données CGNS représentant les résultats d'une simulation numérique est composée d'un ensemble d'entités contiguës de type Zone_t, représentant des domaines de calcul.
Une zone n'est pas nécessairement définie sur une seule entité géométrique. Souvent, on a des zones définies sur plusieurs entités géométriques. De plus, sur une entité géométrique donnée, plusieurs zones peuvent cohabiter.
Étant donné le fait qu'on utilise des surfaces définies géométriquement, leur maillage peut être contenu dans plusieurs domaines de calcul (zones en CGNS). Donc, il faut récupérer les solutions qui les concernent.

Il y a trois façons de définir un maillage sur une face définie géométriquement :
– le maillage d'un domaine de calcul (zone en CGNS) coïncide topologiquement

avec la géométrie, et donc il est entièrement défini dans la zone concernée ;

- si la face représente la frontière entre deux ou plusieurs domaines de calculs (zones en CGNS), elle peut être récupérée en passant par la connectivité entre ces zones ;
- si la face coïncide avec une des frontières du système, elle peut être repérée en utilisant les conditions frontières.

Dans le cas où le rendement est calculé entre l'entrée et la sortie du système, les faces d'entrée et de sortie doivent être représentées par les conditions frontière. De plus, les conditions frontière sont bien identifiées : l'entrée est définie dans la zone CGNS où se trouve sous le noeud de type *ZoneBC* correspondant une condition frontière (*BC*) de type *BCInflow*, et la sortie est définie dans la zone qui se trouve sous le même noeud *ZoneBC* avec une condition frontière (*BC*) de type *BCOutflow*. La turbine étant en rotation, le fichier CGNS contient un noeud de type *RotatingCoordinates* qui définit les coordonnées de rotation, comme le centre de rotation et le vecteur de rotation. Donc, la tâche de retrouver la surface d'entrée et celle de sortie et la surface de la pale est de beaucoup simplifiée.

Le façon que nous avons choisi pour retrouver le maillage, et implicitement les solutions calculées sur ce maillage, est basé sur XML. Nous avons choisi d'utiliser le format XML, car il nous donne la possibilité de définir de l'extérieur les entités CGNS dont nous avons besoin dans le but de calculer le rendement.

Les données récupérées sont envoyées à travers une interface (qui sépare le système de manipulation des données de la partie qui fait le traitement de données) à la bibliothèque ou au logiciel qui fait le calcul du rendement.

Cette architecture nous donne la flexibilité d'avoir des *« plug-in »* sans être obligé de changer le gestionnaire XML de la base de données CGNS.

Quatre nouvelles balises de commande XML ont été introduites à ce niveau :

- < OPALE.compute type= "rendement" >, qui spécifie au système qu'il s'agit d'une séquence de commande XML et qu'il faut passer en mode calcul. L'attribut *type* a été introduit pour pouvoir spécifier différents types de calcule.

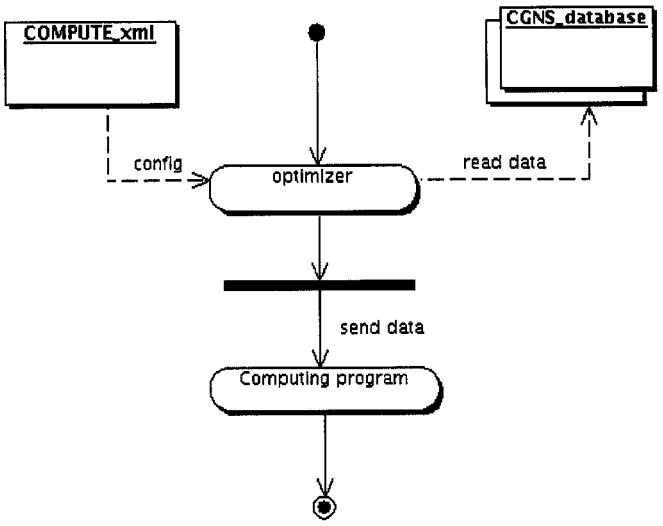


FIGURE 5.4 Le processus de calcul du rendement basé sur CGNS

- < OPALE.input >, qui spécifie qu'il s'agit de la face d'entrée,
- < OPALE.blade >, qui spécifie qu'il s'agit de la pale,
- < OPALE.output>, qui spécifie qu'il s'agit de la face de sortie.

Toutes les zones CGNS comprises entre une balise de départ et la balise de fin correspondante définissent une surface.

Au travers de l'interface graphique et à partir de la structure XML du fichier CGNS, nous pouvons générer un fichier de calcul du rendement en spécifiant les trois zones concernées : l'entrée, l'aube et la sortie. Les balises de configuration sont ajoutées de façon automatique dans le fichier de calcul.

Nous avons calculé les pertes d'une turbine hydraulique en utilisant une base de données CGNS produite par le résoluteur NUMECA. En utilisant le logiciel développé, la base de données CGNS a été enrichie et modifiée pour qu'elle corresponde au modèle de données proposé. Le rendement a été calculé entre l'entrée et la sortie du domaine de calcul. Les résultats que nous avons obtenus étant en concordance avec des résultats obtenus par des autres moyens de calcul du rendement, qu'ils soient numériques ou non, valident tant l'approche que nous avons proposée (le modèle de données), que le fait que de nouvelles disciplines comme la simulation numérique

à haute fidélité peuvent et seront intégrées dans le processus de design. La performance des outils développés et l'exactitude des modèles numériques donnent des résultats qui peuvent être comparés avec les résultats expérimentaux, ce qui permet de valider et d'imposer ces nouvelles disciplines.

La figure montre l'interface developpé pour générer le fichier de calcul.

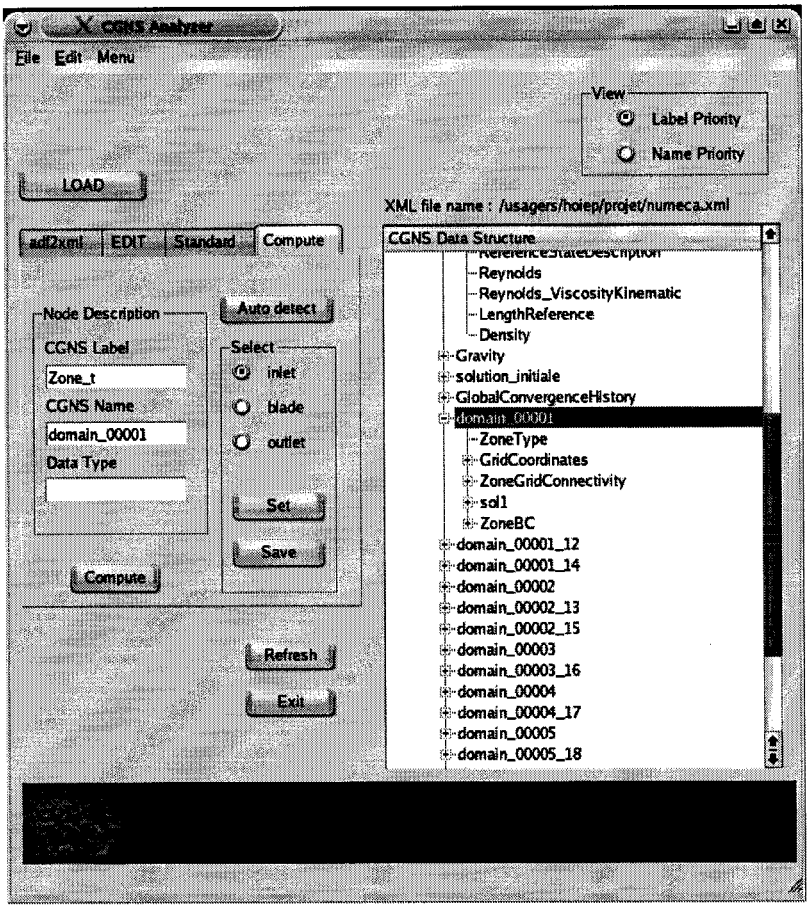


FIGURE 5.5 L'interface graphique qui permettre de générer le fichier de calcul

CONCLUSION

Le but de ce projet était de proposer un modèle de représentation des informations numériques pour l'analyse numérique et l'optimisation multidisciplinaire, incluant à la fois la CFD et la CAO.
Le modèle développé devait assurer l'échange des données CFD entre des applications et des outils d'analyse et de design, commerciaux ou *«maison»*, ce qui nous a fourni un cadre où il était nécessaire de considérer les activités d'analyse et de design dans un cycle itératif d'optimisation. Afin de choisir l'ensemble de technologies les plus appropriées pour la construction du modèle, nous avons évaluer les approches existantes du point de vue de l'analyse numérique et de l'optimisation multidisciplinaire et proposé un modèle qui combine ces approches.

Le projet étant développé en collaboration avec l'IREQ (l'Institut de Recherche d'Hydro-Québec), le modèle de données CFD développé avait comme but de fournir l'infrastructure pour représenter des informations numériques dans un domaine bien particulier, celui des turbines hydrauliques. L'idée derrière ce projet était de développer une méthode d'optimisation des aubes de turbines hydrauliques. L'optimisation de la géométrie des pales devait se faire en cherchant à maximiser le rendement des turbines hydrauliques. Pour que le processus d'optimisation soit efficace, les composantes du processus devaient être intégrées dans une boucle itérative et automatisée.

Une étape importante dans un schéma classique d'automatisation d'un processus complexe composé lui-même de plusieurs sous-processus est d'assurer la compatibilité et la consistance des données échangées par ces sous-processus.

Dans notre cas, pour pouvoir automatiser le processus d'optimisation des pales d'une turbine hydraulique, il fallait assurer la compatibilité et la consistance des données

produites par différents résoluteurs et utilisées par les outils de calcul de rendement.

Plusieurs résoluteurs, parmi lesquels on peut mentionner CFX et NUMECA, ont la capacité de sauvegarder les résultats d'une simulation numérique dans le format CGNS et il existe aussi des traducteurs *«maison»* qui traduisent en format CGNS un fichier de données issues d'une simulation numérique avec des résoluteurs comme TaskFlow. La comparaison structurale de ces bases de données CGNS a mis en évidence qu'il y a encore de très grandes différences au niveau de la structure et du contenu de ces données selon l'application qui les a générées même si conceptuellement les données qui y sont stockées sont «compatibles».
Pour qu'elles puissent être utilisées dans un processus d'analyse automatisé, nous sommes arrivés à la conclusion que les bases de données générées de façon automatique par les différents résoluteurs devaient être converties en un format «standard». Deux étapes devaient être accomplies pour constituer l'infrastructure nécessaire pour assurer l'échange propre et sans perte de données :

- définir un modèle de représentation de données qui donne le cadre nécessaire pour représenter les étapes de design dans une boucle automatisée ;
- trouver une façon efficace d'implanter ce modèle.

À la suite d'une brève évaluation des approches de représentation des informations du point de vue de l'analyse numérique, nous avons constaté que :

- toutes les normes sont structurées sensiblement de la même manière [voir figure 4.1] ;
- toutes les normes essayent de définir des formats neutres (donc, avec un minimum de contraintes) ;
- la tâche d'implanter les contraintes spécifiques à un domaine particulier est laissée aux applications ;

- l'échange de données entre les outils d'analyse et d'optimisation est accompli en utilisant des traducteurs d'un format de représentation à l'autre.

Les problèmes de base avec les standards basés sur des formats neutres, comme CGNS, STEP et IGES, sont la duplication des données, l'ambiguïté de l'information dues au modèle de données très général et l'impossibilité d'imposer des contraintes spécifiques à un domaine d'application donné.

Il n'existe pas de modèle de données bien défini tant du point de vue structural que comportemental spécifique à un domaine particulier d'application CFD. Cela provoque une incompatibilité et une inconsistance des données lors de la phase d'échange. Nous avons donc défini théoriquement un modèle de données «standard», pour notre domaine d'application qui représente en fait un sous-ensemble du méta-modèle global CFD.

Le modèle de données CFD «standard» a été développé en se basant sur la couche abstraite (*SIDS*) de la norme CGNS. Nous avons choisi de l'utiliser car nous avons constaté qu'elle définit le plus complètement le méta-modèle abstrait de données CFD. De plus, la norme CGNS fera partie de la norme STEP en ce qui concerne la représentation des données CFD. Nous avons utilisé la couche persistance de la norme CGNS pour stocker les données décrites par le modèle OPALE.

Le modèle de données «standard» lui-même a été exprimé en utilisant le langage de balises XML. En effet, du point de vue de l'implantation physique du modèle de données, nous voulions qu'il soit facile d'implanter les fonctionnalités suivantes :

- disposer d'un mécanisme efficace pour vérifier et contrôler la façon dont les résultats d'une simulation CFD sont structurés dans la base de données ;
- vérifier que la base de données CFD soit complètement définie avant de l'utiliser dans un processus d'optimisation ou d'analyse numérique ;
- pouvoir facilement modifier la structure de la base de données en fonction des

besoins spécifiques aux applications d'analyse ;
- accéder facilement aux entités géométriques sur lesquelles le maillage est défini.

Grâce aux mécanismes dont XML dispose, toutes les fonctionnalités ont pu être implantées. Le modèle a été exprimé en utilisant un dictionnaire de données (DTD).

Il restait à trouver une façon d'implanter ce sous-ensemble en sachant que les normes existantes n'implantent que le méta-modèle au complet.
Nous proposons et présentons au chapitre 4 une façon de remplacer la structure en trois couches, implantée par les normes actuelles, par une structure en quatre couches. Nous avons conclu de l'analyse des approches existantes que l'architecture est incomplète et qu'elle doit intégrer une couche supplémentaire au niveau de la couche intermédiaire ; la couche grammaire [voir figure 4.2]. Celle-ci a comme but de donner le cadre nécessaire pour pouvoir définir des sous-ensembles de contraintes spécifiques aux domaines d'applications particuliers. En passant par cette couche intermédiaire, qui définit des contraintes et des comportements spécifiques, nous sommes assurés que la structure de données est complètement définie. Par ailleurs, la couche de grammaire permet de donner aux applications un niveau de connaissance minimal pour assurer un échange de données propre et sans perte. Avec une grammaire, nous sommes sûrs qu'il n'y aura pas de données corrompues lors d'échanges de données si chaque participant connaît le dictionnaire de données, donc la structure à laquelle les données doivent absolument se conformer.

Nous avons développé une méthode qui, nous permet d'amener la base de données CGNS dans la forme «standard» définie a priori en passant la structure d'une base de données à un processus de validation et filtrage. La méthode développée comprend les étapes de validation, filtrage, et ajout d'informations manquantes qui sont executées sur un fichier XML décrivant la structure de la base de données CGNS.

Elle s'avère très efficace car elle ne travaille que sur la structure extraite de la base de données CGNS exprimée en XML, et non pas directement sur la base de données. L'approche mise au point permet d'envisager plusieurs applications ultérieures. Tout d'abord, la méthode est suffisamment générale pour pouvoir être étendue à d'autres structures de stockage de données CFD. Tout en demeurant dans le domaine de l'analyse et de l'optimisation multidisciplinaire, la méthode pourrait être étendue à la CAO. Si on reprend la façon de définir et de représenter des modèles de données pour le domaine de la CAO, la même approche permettrait la définition d'un modèle de données géométriques. De plus, les methodes de représentation des données CFD et CAO seraient uniformes ce qui simplifierait le traitement et la validation globale des données.

Travaux futurs

Dans la prochaine étape, il faut passer du DTD au XSD en ce qui concerne l'implantation de la couche grammaire. Avec le XSD, on peut employer le concept d'héritage, ce qui facilitera l'implantation des entités ayant un comportement semblable comme par exemple les zones structurées et les zones non-structurées. Un autre avantage du XSD est le fait qu'on peut définir des types de données. De plus, avec le DTD, il faut inscrire les balises XML dans l'ordre dans lequel elles ont été déclarées dans le dictionnaire, ce qui n'est pas le cas avec un XSD.

Ensuite, il faut introduire l'espace de noms, ce qui facilitera la différentiation des types de donnés. En utilisant un espace de noms, seuls les types de données concernés seront analysés, ce qui augmentera l'efficacité des opérations d'entrées/sortie. Par exemple, on peut avoir plusieurs entités de type FlowSolutions dans un même fichier dans le cas d'un calcul, ou on peut avoir des ensembles de type ReferenceState (définis pour chaque type de solveur) dans le même fichier.

Une autre étape sera de définir le dictionnaire et la grammaire du meta-modèle

CFD et CAO en XML. Par la suite, en utilisant l'héritage implanté par XSD, plus le concept d'espace de noms, on pourra définir des sous ensembles de restrictions et de règles de comportement (grammaires) spécifiques à chaque domaine d'application. Le résultat sera un meta-modèle de données très flexible, très général, mais très facile à contraindre.

Le concept peut être facilement utilisé pour faire des conversions de données d'un format à l'autre.

Un autre avantage d'un modèle développé avec DTD/XSD + XML est le fait qu'il est indépendant de la platforme et qu'il est auto-validable.

Pour que l'approche soit complètement utilisable, il faut définir la couche abstraite pour la CAO. Le but étant d'avoir un système unifié qui permette de considérer les étapes de design dans une boucle automatisée, le lien avec la géométrie est nécessaire. Le SIDS donne déjà la base pour intégrer la géométrie. En définissant la couche grammaire pour la CAO de la même façon, et en utilisant XML, on introduit un dégrée d'abstraction qui nous donnera la possibilité d'utiliser plusieurs formats de représentation de la géométrie.

En ce qui concerne le format de représentation des données CAO recommandé, le sujet est délicat. STEP est la norme qui s'impose du point de vue de la maturité à définir le modèle abstraite. En ce qui concerne l'implémentation physique du modèle, les opinions sont divisées. STEP utilise son langage EXPRESS pour définir les entités géométriques, qui comme nous l'avons déjà mentionné, est assez difficilement interprétable et les modèles des données STEP décrits en XML sont de plus en plus utilisées. Toutes les opérations, contraintes structurales ou de comportement, qui peuvent être définis à l'aide d'EXPRESS, peuvent également être décrites par XML (XSD), qui est beaucoup plus clair et facile à utiliser.

Donc, l'approche proposée est de définir le meta-modèle CAO à l'aide de la couche abstraite de STEP, et de laisser un certaine dégré de flexibilité quant à l'implémentation physique du modèle (STEP, XML, VRML, etc.).

Il faut tenir compte du fait que la norme CGNS sera bientôt intégrée dans STEP comme standard pour représenter les données CFD. De plus, si l'on a défini la géométrie à l'aide de XML, l'utilisation de xlink pour assurer le liens entre le maillage et la géométrie dont on a besoin est une solution qui s'impose d'elle même.
Un autre aspect important qu'il faut explorer est la façon dont on peut forcer les applications tant commerciales que *« maison »* à utiliser la couche grammaire pour accomplir les opérations d'entrée/sortie. Il est très important que les applications connaissent la couche grammaire si on veut vraiment standardiser l'échange des données. À la limite il faut faire passer les opérations d'entrée/sortie par un outil de mise en forme qui lui connaît la couche grammaire.

RÉFÉRENCES

(2000). Iges project. http ://www.nist.gov/iges.

(2000). Step tools, inc. http ://www.steptools.com.

(2000). Vrml site. http ://www.vrml.com/.

(2001). The CFD general notation system. http ://www.cgns.org.

(2001). Prostep site. http ://www.prostep.com/de/.

(2001). Sids on linedocumentation. http ://www.grc.nasa.gov/WWW/cgns/sids.

(2001). XML cover pages. http ://xml.coverpages.org/stepExpressXML.html.

(2001). Xvl site. http ://www.xvl.com/.

(2002). Document type definition (DTD) site. http ://www.w3.org/XML/dtd.

(2002). Extensible markup language. http ://www.w3.org/XML.

(2002). Step on line documentation. http ://www.mel.nist.gov/sc4/www/stepdocs.htm.

(2002). Xml schema definition (XSD) site. http ://www.w3.org/XML/Schema.

BARKMEYER, E. J. et LUBELL, J. (2001). Xml representation of express models and data. http ://www.mel.nist.gov/msidlibrary/doc/xse2001.pdf.

BASU, D. et KUMAR, S. (1995). Importing mesh entities through iges/pdes. *Advances in Engineering Software*, *23*, 151–161.

BENYO, T. L. (2002). Project integration architecture (PIA) and computational analysis programming interface (CAPRI) for accessing geometry data from CAD files. *Proceeding of the 40th AIAA Aerospace Sciences Meeting & Exhibit.* Reno, NV.

MANGESH, B., BLAIR, D. et HARDWICK, M. (2000). Migrating from iges to step : one to one translation of iges drawing to step drafting data. *Computers in Industry, 41*, 261–277.

POIRIER, D., ALLMARAS, STEVEN, R., DOUGLAS, R., M., SMITH, MATTHEW, F. et ENOMOTO, FRANCISC, Y. (1998). The cgns system. *AIAA-98-3007.*

ANNEXE I

I.1 Cas test et utilisation

La figure montre le diagramme de contexte de l'Analyseur et Standardiseur CGNS. On y identifie clairement, quatre cas d'utilisation :

- Extraire structure
- Verifier conformité
- Editer
- Standardiser

le fichier CGNS. La hiérarchie ou le interdépendence de ces cas d'utilisation est aussi clairement figurée sur le diagramme.

Le premiere cas d'utilisation, Extraire structure (voir le tableau 1.2) correspond à la situation où l'utilisateur n'envisage que d'extraire, en format XML, la structure d'une base de données CGNS dans le but de l'analyser ultérieurement. Il y a plusieurs outils xml permettant d'effectuer des analyses sur un fichier xml.

Le deuxiéme cas d'utilisation, Verifier la conformité (voir le tableau 1.2) correspond à la situation où l'utilisateur veut vérifier si une base de données CGNS est, ou non, conforme à un modèle donné. À ce niveau l'utilisateur a deux options : vérifier si le fichier CGNS est conforme à la norme (le cas le plus general - SIDS), ou vérifier si le fichier CGNS vérifie un ensemble des conditions imposeés par l'utilisateur (le cas d'OPALE) qui, comme le diagramme montre, est un sous cas du cas général (SIDS). Ce cas d'utilisation inclut le cas précedent Extraire Structure.

Le troisième cas d'utilisation, Editer (voir le tableau 1.3), inclut les deux cas précedents Extraire structure et Vérifie confomité. Dans ce cas, l'utilisateur modifie la

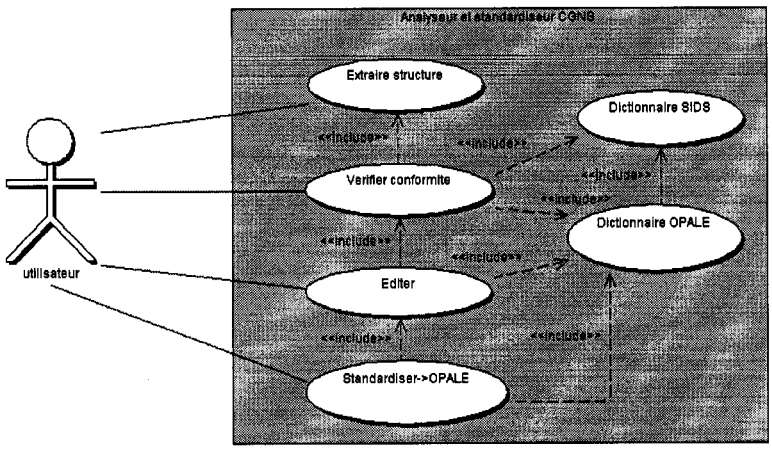


FIGURE I.1 Cas d'utilisation

structure des données cgns decrite en XML.

Le quatriemme cas d'utilisation, Standardiser->OPALE (voir le tableau 1.4), qui est le cas principal, inclut les trois cas précedents Verifier la conformité, Editer et Extraire structure. Si l'utilisateur a choisi ce cas d'utilisation, il veut mettre dans la forme standard une base de données CGNS. Le résulat de cette action sera un nouveau fichier CGNS.

Cas d'utilisation : *Extraire structure*	
Acteur	Utilisateur
Description	L'utilisateur envisage d'extraire la structure d'un fichier CGNS en format XML. Il doit sélectioner le fichier CGNS à analyser et aussi le niveau d'information qu'il veut extraire. Présentement trois niveaux d'information sont disponibles à l'utilisateur : - *Label* - *Name* - *Type*

Cas d'utilisation : *Verifier la conformité*	
Acteur	Utilisateur
Description	L'utilisateur envisage de vérifier la conformité d'un fichier CGNS. A travers d'une interface graphique, il selectionne le fichier CGNS à analyser et aussi le dictionaire à utiliser.

Cas d'utilisation : *Editer*	
Acteur	Utilisateur
Description	L'utilisateur envisage de modifier la structure des données cgns decrite en XML dans le but de standardiser le fichier CGNS. À ce niveau, la structure du fichier cgns est montré à l'utilisateur a travers de cette interface graphique. S'il s'agit des modifications punctuelles comme par exemple modifier le nom d'une entité, ou ajouter une entité simple, les modifications peuvent être faite directement à travers de cette interface. S'il faut ajouter beaucoup plus des données, un fichier contenant les données sera indique au logiciel. Ce cas d'utilisation inclut les deux cas précedents *Extraire structure* et *Vérifie confomité* .

Cas d'utilisation : *Standardiser -> OPALE*	
Acteur	Utilisateur
Description	L'utilisateur envisage de mettre dans la forme standard une base de données CGNS. L'utilisateur indique le fichier à standardiser. Le logiciel lui demande le dictionnaire de données à utiliser. S'il y a des erreurs ou des non concordances, elles seront sous lignées sur la structure du fichier cgns montrée. Par suite, les etapes decrits dans le paragraph *Editer* doivent être suivi. Ce cas inclut les trois cas précedents *Verifier la conformité, Editer et Extraire structure* . Le résulat de cette action sera un nouveau fichier CGNS.

Une version d'analyseur des structures CGNS peut être celle montrée dans la figure suivante. Dans la partie gauche on retrouvent les trois etapes de la diagramme d'activites OPALE (Extraire la structure, Editer la structure et Mise en forme standard du fichier CGNS).

I.2 Optimisation d'un fichier des solutions CGNS, produit par CFX

L'utilisateur a simulé un écoulement dans une roue de turbine hydraulique en utilisant le logiciel CFX5. Il a sauvegardé les résultats de la simulation dans le format cgns : roue.cgns.

Ensuite, il utilise le logiciel d'optimisation décrit ici, pour restructurer et optimiser le résultat de cette simulation dans le but d'integrer les résultats de la simulation dans un processus d'optimisation automatisé de la rue, en produisant une nouvelle base de données roue_opt.cgns.

Le logiciel vérifie si le fichier roue.cgns est conforme au modèle de données décrit dans le fichier OPALE.dtd. Il constate que le fichier est non-conforme et il demande

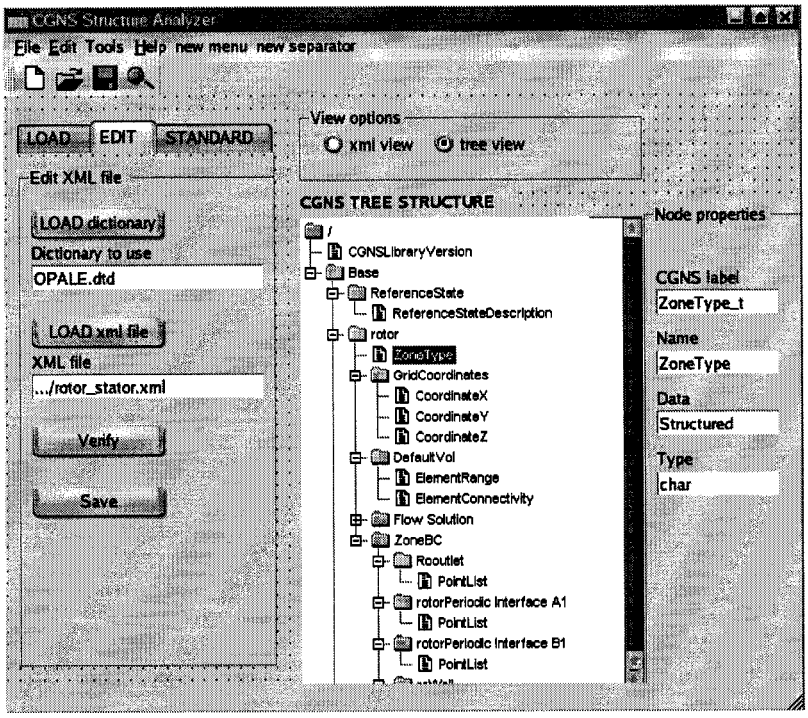


FIGURE I.2 L'interface graphique

à l'utilisateur le fichier filtre et le fichier de données à ajouter.

Le logiciel est alors en mesure de restructurer les données.

Ensuite, il demande à l'utilisateur de choisir s'il veut sauvegarder la nouvelle base de données dans le même fichier cgns (roue.cgns) ou de créer un nouveau fichier cgns.

Réponse : dans la même base.

Le gogiciel écrit la structure de la nouvelle base de données dans le même fichier cgns.

Il demande ensuite si la structure xml optimisée doit être stockée.

Réponse : oui.

Il demande si l'utilisateur veut ajouter le fichier xml contenant la structure dans le même fichier avec la base de données ou non.

Réponse : oui.

Le fichier xml contenant la structure est sauvegarder dans le fichier roue.cgns.

A la fin, l'utilisateur se trouvent avec un fichier cgns contenant la base de données

cgns originale, la base de données optimisée et un fichier xml décrivant la structure optimisée de cette base de données.

ANNEXE II

II.1 La structure d'un fichier des donnees IGES

CFX-Build 5 generated IGES File from rotorNew database S0000001
/home/cerca/segal/iepan/cfx/test_IGES/iges.db S0000002
„22HMSC.Patran IGES Access,47H/home/cerca/segal/iepan G0000001
1.0, 6,1HM,1,0,13H030820.154211,0.500000E-02,1.05, G0000004
5Hiepan,14HCFX-Build User, G0000005
9,0; G0000006

110 11 0 2 10 0 0 0 001010000D0000063
110 0 0 2 0 0 0LINE 1D0000064
122 115 2 1 0 0 0 001010000D0000065
122 0 0 1 0 0 0 TABCYL 1D0000066
112 116 2 1 0 0 0 001010000D0000067
112 0 0 10 0 0 0SPLINE 1D0000068
110 0 0 2 0 0 0LINE 2D0000096
116, 0.421743002E-02,-0.252743363E+00, 0.100000000E+01; 25P0000013
116, 0.528298989E-01,-0.267197281E+00, 0.100000000E+01; 27P0000014
0.528298989E-01,-0.267197281E+00, 0.100000000E+01, 139P0000864
0.000000000E+00, 0.100000000E+01, 0.000000000E+00, 139P0000865
0.100000000E+01; 139P0000866
S 2G 6D 140P 866 T0000001

II.2 La structure d'un fichier des donnees PIRATE

```
// Maillage provenant du reordonnement des sommets
// du fichier all_layer.pie

Champ<double> coordonneesSommets() = {
-2.0000000000e+01 -2.0000000000e+01
-1.0988282000e+01 -2.0000000000e+01
-9.8007538000e-01 -2.0000000000e+01
1.0426306000e+01 -2.0000000000e+01
2.0000000000e+01 -2.0000000000e+01
};

Champ<int> codesSommets() = {
1 12 12 12 2 23 23 23 23 3
34 34 34 34 4 41 41 41 41 5
56 56 56 56 56 56 56 56 56 56
56 56 56 56 56 56 56 56 56 56
56 56 56 56 56 56 56 56 56 56
101 101 101 101 101 101 101 101 101 101
101 101 101 101 101 101 101 101 101 101
};

Champ<int> connecTriangles() = {
471 456 479
448 457 451
2682 2667 4
470 499 476
178 179 6470
```

```
179 6479 6470
6479 6268 6259
};

Maillage meshRae( GeoRae, coordonneesSommets%2, codesSommets%1 ) = {
zone zone_triangles ( LagrTrian03, coordonneesSommets%2, connecTriangles%3,
101, codesSommets%1 );
};

// Solution restart provenant de la traduction du fichier restart.out
// traduit dans le format .pie
//
// nnode = 6480 (Nombre de sommets)


// Champ rho de la densite
Champ<double> rho() = {
1.000450006E+00
1.000963981E+00
1.002843868E+00
1.001216851E+00
};

// Champ rho_u de la vitesse en x
Champ<double> rho_u() = {
8.625313203E-01
8.620446330E-01
8.623653742E-01
8.619244851E-01
};
```

// Champ rho_v de la vitesse en y
Champ<double> rho_v() = {
3.708491238E-02
3.720511950E-02
3.521828758E-02
3.514636104E-02
};

// Champ rho_E
Champ<double> rho_E() = {
2.874142823E+00
2.875413695E+00
2.880060709E+00
};

// Champ Mach
Champ<double> Mach() = {
7.292418378E-01
7.283751521E-01
7.271460004E-01
7.279785846E-01
};

// Champ turbulent1
Champ<double> turbulent1() = {
1.000000000E-02
1.000000000E-02
1.000000000E-02
1.000000000E-02

};

solution restart(meshRae) = {
variable rho (LagrTrian03, rho, connecTriangles%3, zone_triangles);
variable rho_u (LagrTrian03, rho_u, connecTriangles%3, zone_triangles);
variable rho_v (LagrTrian03, rho_v, connecTriangles%3, zone_triangles);
variable rho_E (LagrTrian03, rho_E, connecTriangles%3, zone_triangles);
variable press (LagrTrian03, press, connecTriangles%3, zone_triangles);
variable Mach (LagrTrian03, Mach , connecTriangles%3, zone_triangles);
variable Spalart_Allmaras (LagrTrian03, turbulent1, connecTriangles%3, zone_triangles
);
};

II.3 La structure d'un fichier des donnees *referenceState.xml* et *flowEquationSet.xml*

<?xml version="1.0" ?>

<CGNS-Tree-t name="numeca" OPENMode="READWRITE" >

<CGNSBase-t name="base_1" physicalDimension="3" cellDimension="3">

<addStructure-t name="numeca_ReferenceState">
<UserDefinedData-t name="ComputationParameters">
<DataArray-t name="AUTO_GRID_MESH" type="C1" >yes</DataArray-t>
<DataArray-t name="COARSE_FLAG" type="C1" >111</DataArray-t>
<DataArray-t name="NUMBER_OF_GRID_LEVEL" type="R8" >3</DataArray-
t>
<DataArray-t name="VERSION" type="R8" >4</DataArray-t>

<DataArray-t name="NAME" type="C1" >niveau_1</DataArray-t>
<DataArray-t name="FLUID_NAME" type="C1" >WATER</DataArray-t>
<DataArray-t name="FLUID_TYPE" type="C1" >Incompressible</DataArray-t>
<DataArray-t name="AUTO_GRID_MESH" type="C1" >yes</DataArray-t>
<DataArray-t name="START_FROM_INITIAL_SOLUTION_FILE" type="C1" >no</DataArray-t>
<DataArray-t name="BETPAR" type="R8" >3</DataArray-t>
<DataArray-t name="OMGSY" type="R8" >0</DataArray-t>
<DataArray-t name="OMGSY" type="R8" >0</DataArray-t>
<DataArray-t name="NSTAGE" type="R8" >4</DataArray-t>
<DataArray-t name="NGRID" type="R8" >3</DataArray-t>
<DataArray-t name="KEGRID" type="R8" >2</DataArray-t>
<DataArray-t name="AUTO_GRID_MESH" type="C1" >yes</DataArray-t>
</UserDefinedData-t>

<ReferenceState-t name="ReferenceState" referenceStateDescription="Export mesh and solution data from CFX5">
<DataArray-t name="Reynolds" type="R8" >2.66e+006</DataArray-t>
<DataArray-t name="Reynolds_Viscosity" type="R8" >3.28947e-007</DataArray-t>
<DataArray-t name="Reynolds_Velocity" type="R8" >0.875</DataArray-t>
<DataArray-t name="Reynolds_Length" type="R8" > 1 </DataArray-t>
<DataArray-t name="Temperature" type="I4" > 288 </DataArray-t>
<DataArray-t name="Density" type="I4" > 988 </DataArray-t>
</ReferenceState-t> <!- fin du ReferenceState ->

<Gravity-t name="Gravity" i="0" j="0" k="-1" > </Gravity-t>

<SimulationType-t name="SimulationType">NonTimeAccurate</SimulationType-t>
<Descriptor-t name="Descriptor">totoDescriptor</Descriptor-t>
<DataClass-t name="DataClass" >Dimensional</DataClass-t>
<DimensionalUnits-t name="DimensionalUnits" mass="Kilogram" length="Meter" time="Second" temperature="Kelvin" angle="Degree"> </DimensionalUnits-t>
<FlowEquationSet-t name="FlowEquationSet" equationDimension="3" >
<GoverningEquations-t name="GoverningEquations">NSTurbulentIncompressible</Governi
t>
<GasModel-t name="GasModel">CaloricallyPerfect</GasModel-t>
<ViscosityModel-t name="ViscosityModel" modelType="Constant">
<DataArray-t name="ViscosityMolecular" type="R8" >0.00100</DataArray-t>
<DataArray-t name="TemperatureReference" type="R8" >293.16 </DataArray-t>
</ViscosityModel-t>
<TurbulenceClosure-t name="EddyViscosity">EddyViscosity</TurbulenceClosure-t>
<TurbulenceModel-t name="TModel" modelType="TwoEquation_Wilcox" >
<DiffusionModel-t name="3D">1,1,1,1,1,1</DiffusionModel-t>
<DataClass-t name="DataClass" >NondimensionalParameter</DataClass-t>
<DataArray-t name="TurbulentKe_Cmu" type="R8" >0.09</DataArray-t>
<DataArray-t name="TurbulentKe_Ce1" type="R8" >1.44</DataArray-t>
<DataArray-t name="TurbulentKe_Ce2" type="R8" >1.92</DataArray-t>
<DataArray-t name="TurbulentKe_SigmaE" type="R8" >1.3 </DataArray-t>
<DataArray-t name="TurbulentKe_SigmaK" type="R8" >1 </DataArray-t>
<DataArray-t name="TurbulentKe_SigmaRho" type="R8" >0.5 </DataArray-t>
</TurbulenceModel-t>

</FlowEquationSet-t>
<ConvergenceHistory-t name="CoHist" iterations="100" normDefinitions="str1 str2 str3 ..."> </ConvergenceHistory-t>

<Family-t name="Family1">
<GeometryFormat-t name="GFo" >PIRATE</GeometryFormat-t>
<GeometryFile-t name="GFi" >FILE_TEST_1</GeometryFile-t>
<GeometryEntity-t name="GEn"> FACE_ENTREE</GeometryEntity-t>
</Family-t>
</addStructure-t>
</CGNSBase-t>
</CGNS-Tree-t>

II.4 Le dictionnaire de données d'OPALE (OPALE.dtd)

<!ENTITY % modele.strings SYSTEM "modele-strings.dtd" >
%modele.strings;
<!- ================================= ->
<!ELEMENT OPALE.tree (CGNSBase_t+)>
<!ATTLIST OPALE.tree name CDATA #IMPLIED >
<!- ================================= ->
<!ELEMENT CGNSBase_t (DataClass_t,
DimensionalUnits_t,
ReferenceState_t,
FlowEquationSet_t,
Zone_t+,
Gravity_t,
Family_t+,

```
Descriptor_t+,
ConvergenceHistory_t,
SimulationType_t )>
<!ATTLIST CGNSBase_t name CDATA #REQUIRED >
<!-                                  ->
<!ELEMENT ReferenceState_t ( Descriptor_t+,
DataArray_t+,
UserDefinedData_t? )>
<!ATTLIST ReferenceState_t name %ReferenceState_t.s; #REQUIRED>
<!-============================================= ->
<!ELEMENT Zone_t ( ZoneType_t,
GridCoordinates_t*,
Elements_t*,
FlowSolution_t?,
ZoneGridConnectivity_t?,
Family_t*,
ZoneBC_t? )>
<!ATTLIST Zone_t name CDATA #REQUIRED >
<!-============================================= ->
<!ELEMENT ZoneType_t (#PCDATA) >
<!ATTLIST ZoneType_t name %ZoneType_t.s; #REQUIRED >
<!-============================================= ->
<!ELEMENT GridLocation_t (#PCDATA) >
<!ATTLIST GridLocation_t name %GridLocation_t.s; #REQUIRED >
<!-============================================= ->
<!ELEMENT GridCoordinates_t ( DataArray_t? )>
<!ATTLIST GridCoordinates_t name %GridCoordinates_t.s; #REQUIRED >
<!-============================================= ->
```

```
<!ELEMENT SimulationType_t (#PCDATA )>
<!ATTLIST SimulationType_t name %SimulationType_t.s; #REQUIRED >
<!-============================================= ->
<!ELEMENT Descriptor_t (#PCDATA )>
<!ATTLIST Descriptor_t name CDATA #REQUIRED >
<!-============================================= ->
<!ELEMENT Elements_t ( ElementConnectivity_t?,
IndexRange_t?,
ElementType_t? )>
<!ATTLIST Elements_t name %Elements_t.s; #REQUIRED >
<!-============================================= ->
<!ELEMENT ElementConnectivity_t (#PCDATA )>
<!ATTLIST ElementConnectivity_t name %ElementConnectivity_t.s; #REQUI-
RED >
<!-============================================= ->
<!ELEMENT ElementType_t (#PCDATA )>
<!ATTLIST ElementType_t name %ElementType_t.s; #REQUIRED >
<!-============================================= ->
<!ELEMENT FlowSolution_t ( GridLocation_t,
DataArray_t+ )>
<!ATTLIST FlowSolution_t name %FlowSolution_t.s; #REQUIRED >
<!-============================================= ->
<!ELEMENT Gravity_t ( DataArray? ) >
<!ATTLIST Gravity_t name %Gravity_t.s; #REQUIRED
i CDATA #IMPLIED
j CDATA #IMPLIED
k CDATA #IMPLIED >
<!- ============================================= ->
```

<!ELEMENT FlowEquationSet_t (EquationDimension_t,
GoverningEquations_t,
ThermalConductivityModel_t,
GasModel_t,
ViscosityModel_t,
TurbulenceClosure_t,
TurbulenceModel_t,
DataClass_t?,
DimensionalUnits_t?)>
<!ATTLIST FlowEquationSet_t name %FlowEquationSet_t.s; #REQUIRED >
<!-- ->
<!ELEMENT EquationDimension_t (#PCDATA) >
<!ATTLIST EquationDimension_t name %EquationDimension_t.s; #REQUIRED >
<!-- ->
<!ELEMENT GoverningEquations_t (#PCDATA) >
<!ATTLIST GoverningEquations_t name %GoverningEquations_t.s; #REQUIRED>
<!-- ->
<!ELEMENT ThermalConductivityModel_t (#PCDATA)>
<!ATTLIST ThermalConductivityModel_t name %ThermalConductivityModel_t.s; #REQUIRED>
<!-- ->
<!ELEMENT GasModel_t (#PCDATA) >
<!ATTLIST GasModel_t name %GasModel_t.s; #REQUIRED>
<!-- ->
<!ELEMENT ViscosityModel_t (DataArray_t+)>
<!ATTLIST ViscosityModel_t name %ViscosityModel_t.s; #REQUIRED

```
nodeData %ViscosityModelType.s; #IMPLIED >
<!-========================================== ->
<!ELEMENT TurbulenceModel_t ( DiffusionModel_t?,
DataClass_t?,
DataArray_t+,
Descriptor_t* )>
<!ATTLIST TurbulenceModel_t name %TurbulenceModel_t.s; #REQUIRED
nodeData %TurbulenceModelType.s; #IMPLIED > <!-=====================
>
<!ELEMENT TurbulenceClosure_t ( EMPTY )> <!ATTLIST TurbulenceClo-
sure_t name %TurbulenceClosure_t.s; #REQUIRED
nodeData %TurbulenceClosureType.s; #IMPLIED >
<!-========================================== ->
<!ELEMENT ZoneBC_t ( BC_t+,
Descriptor_t? )>
<!ATTLIST ZoneBC_t name %ZoneBC_t.s; #REQUIRED >
<!-========================================== ->
<!ELEMENT BC_t ( (IndexArray_t|IndexRange_t)?,
Descriptor_t*,
FamilyName_t?,
BCProperty_t* )>
<!ATTLIST BC_t name %BC_t.s; #REQUIRED
nodeData %BCType_t.s; #REQUIRED > <!-=========================
>
<!ELEMENT BCProperty_t ( WallFunction_t )>
<!ATTLIST BCProperty_t name %BCProperty.s; #REQUIRED >
<!-========================================== ->
<!ELEMENT WallFunction_t ( WallFunctionType_t )>
```

<!ATTLIST WallFunction_t name %WallFunction_t.s; #REQUIRED >

<!-== ->

<!ELEMENT WallFunctionType_t (#PCDATA)>

<!ATTLIST WallFunctionType_t name %WallFunctionType_t.s; #REQUIRED

nodeData %WallFunctionType.s; #REQUIRED >

<!-== ->

<!ELEMENT UserDefinedData_t (Descriptor_t*,

DimensionalUnits_t?,

DataArray_t?)>

<!ATTLIST UserDefinedData_t name CDATA #IMPLIED>

<!-== ->

<!ELEMENT ZoneGridConnectivity_t (Descriptor_t*,

GridConnectivity1to1_t*,

GridConnectivity_t*)>

<!ATTLIST ZoneGridConnectivity_t name %ZoneGridConnectivity_t.s; #RE-

QUIRED >

<!-== ->

<!ELEMENT GridConnectivity1to1_t (Transform_t,

IndexRange_t,

IndexRangeDonor_t)>

<!ATTLIST GridConnectivity1to1_t name %GridConnectivity1to1_t.s; #RE-

QUIRED

nodeData CDATA #REQUIRED >

<!-== ->

<!ELEMENT IndexArray_t (#PCDATA)>

<!ATTLIST IndexArray_t name %IndexArray_t.s; #REQUIRED >

<!-== ->

<!ELEMENT IndexRange_t (#PCDATA)>

```
<!ATTLIST IndexRange_t name %IndexRange_t.s; #REQUIRED >
<!-============================================  ->
<!ELEMENT Family_t ( FamilyBC_t?,
GeometryReference_t*,
Ordinal_t?,
Descriptor_t*,
UserDefinedData_t* )>
<!ATTLIST Family_t name CDATA #REQUIRED >
<!-============================================  ->
<!ELEMENT FamilyBC_t EMPTY >
<!ATTLIST FamilyBC_t name CDATA #IMPLIED
nodeData %BCType.s; #REQUIRED >
<!-============================================ ->
<!ELEMENT FamilyName_t EMPTY >
<!ATTLIST FamilyName_t nodeData CDATA #IMPLIED >
<!-============================================  ->
<!ELEMENT GeometryReference_t ( Descriptor_t*,
GeometryFile_t,
GeometryFormat_t,
GeometryEntity_t*,
UserDefinedData_t* )>
<!ATTLIST GeometryReference_t name CDATA #REQUIRED >
<!-============================================  ->
<!ELEMENT GeometryFile_t EMPTY >
<!ATTLIST GeometryFile_t name CDATA #REQUIRED
nodeData CDATA #REQUIRED >
<!-============================================  ->
<!ELEMENT GeometryFormat_t EMPTY >
```

```
<!ATTLIST GeometryFormat_t name CDATA #REQUIRED
nodeData CDATA #REQUIRED >
<!-============================================  ->
<!ELEMENT GeometryEntity_t EMPTY >
<!ATTLIST GeometryEntity_t name CDATA #REQUIRED
nodeData CDATA #REQUIRED >
<!-============================================  ->
<!ELEMENT Transform_t ( #PCDATA )>
<!ATTLIST Transform_t name %Transform.s; #REQUIRED >
<!-============================================  ->
<!ELEMENT DataClass_t ( #PCDATA ) >
<!ATTLIST DataClass_t name %DataClass_t.s; #REQUIRED >
<!-============================================  ->
<!ELEMENT DimensionalUnits_t ( #PCDATA )>
<!ATTLIST DimensionalUnits_t name %DimensionalUnits.s; #REQUIRED >
<!-============================================  ->
<!ELEMENT ConvergenceHistory_t ( Descriptor_t,
DataArray_t )>


<!ATTLIST ConvergenceHistory_t name %ConvergenceHistory.s; #REQUIRED
>
```

ANNEXE III

III.1 Cas test CGNS-CFX5

L'arbre CGNS correspondant à un deuxième cas test avec une géométrie plus réaliste, dont le but était de corriger quelques aspects non réalistes du modèle initial, parmi lesquelles on peut énumérer : aubes ayant une épaisseur non nulle, l'espace entre les pales et les surfaces de sortie et d'entrée et l'écoulement incompressible, est montré dans la figure suivante.

Un autre aspect qui a été réglé dans le deuxième cas de test est celui concernant la périodicité. Dans le premier cas-test, on n'avait pas de la périodicité, car la partie concernée à la simulation était celle comprise entre deux pales (ou entre les deux directrices).
Dans la nouvelle géométrie, les pales et respectivement les directrices ont des épaisseurs non nulles. Donc, la partie concernée à la simulation n'est plus celle compris entre deux palles (ou directrices) d'épaisseur nulle, et plutôt la pale (ou la directrice) est contenue, au milieu de la partie concernée à la simulation, ce qui nous a donner la possibilité d'imposer la périodicité sur les faces.

On remarque un manque d'information dans le fichier CGNS sortie par CFX. S'agir du lien avec la géométrie et la description des équations utiliser pour résoudre le problème.

La même méthodologie avec des maillages définis séparément pour chaque composante (le rotor et le stator), plus précisément un fichier des définitions pour chacune a été employée.
Les étapes :

1. Créer séparément les géométries pour les deux composantes respectivement le rotor et le stator. Un fichier de géométrie par composant.
2. Définir les régions 3D et respectivement 2D correspondantes à chaque solide. Les régions 2D correspondantes au stator sont :

   - statBladeWall
   - statLowerWall
   - statUpperWall
   - statPeriodPair
   - statorIn
   - statorOut

   Les régions 2D correspondantes au rotor sont :

   - rotBladeWall
   - rotLowerWall
   - rotUpperWall
   - rotPeriodPair
   - rotorIn
   - rotorOut
3. Définir de façon individuellement les propriétés du maillage et générer le maillage.

   Les paramètres utilisés pour générer les maillages sont :

   - Inflated Boundary - MaximumThickness = 0.03
   - Maximum Edge Length = 0.03 m

- Nombre de couches pour la peau = 8
- Le factor d'expansion géométrique = 1.2
- L'angle intérieur minimum = 2.5 degrées

4. Générer le fichier de définition approprie à chaque composant (rotor.def et stator.def). L'étape suivante consiste en a définir la nouvelle base de données qui servira effectivement au simulation numérique. Les deux maillages crées jusqu'au maintenant sont importées dans la nouvelle base de données.
5. On définit un groupe de maillage Mesh Group nommé rotorStator.
6. On définit le domaine de calcul avec ses propriétés (Fluid Domain).

III.1.1 Pour rotor

Sélectionner le rotor dans le groupe de maillage

- Créer le domaine de calcul pour rotor : Rotor Domain
- Domain Motion = Rotating
- Le type de fluide est l'eau
- Dans les options du domaine (Domain Options) on a :
- La pression de référence est 1 bar,
- La vitesse angulaire est 1.934999 rad/sec, ce qui corresponde à 100 rot/min.
- L'axe de rotation est l'axe Z (Coord 0.3).

III.1.2 Pour stator

Sélectionner le stator dans le groupe de maillage

- Créer le domaine de calcul pour stator : Stator Domain
- Domain Motion = Stationary
- Le type de fluide est l'eau
- Dans les options du domaine (Domain Options) on a la pression de référence á 1 bar.

Une fois les domaines définis, on impose les conditions aux frontières pour le groupe de maillage qu'on vient de créé

III.1.3 Pour l'entrée (Inlet)

On sélect le stator et on lui impose :

- Type = Inlet
- Frame = Stationary
- On sélect la région 2D nommé inflow, comme la région inlet
- Créer Fluid Boundary avec les valeurs :
  - Les composantes cartésiennes de la vélocité
  - U = -0,725* cos(theta+0,6981)[m/s]
  - V = -0,725* sin(theta+0,6981)[m/s]
  - W = 0.0
- La température statique : 288.0 K

III.1.4 Pour sortie (Outlet)

On sélect le rotor et on lui impose :

- Type = Outlet

- Frame = Rotating

III.1.5 Pour Wall BC

Les conditions sont posées séparément pour les deux composantes.

Pour rotor on lui impose le type = Wall et Frame = Rotating (on sélect : BladeWall, statLowerWall et statUpperWall)

Pour stator on lui impose le type = Wall et Frame = Stationary (on sélectÂ?rotBladeWall, rotLowerWall et rotUpperWall)

III.1.6 L'interface entre les solides

Pour définir l'interface entre les deux solides, on utilise l'option Domain Interface :

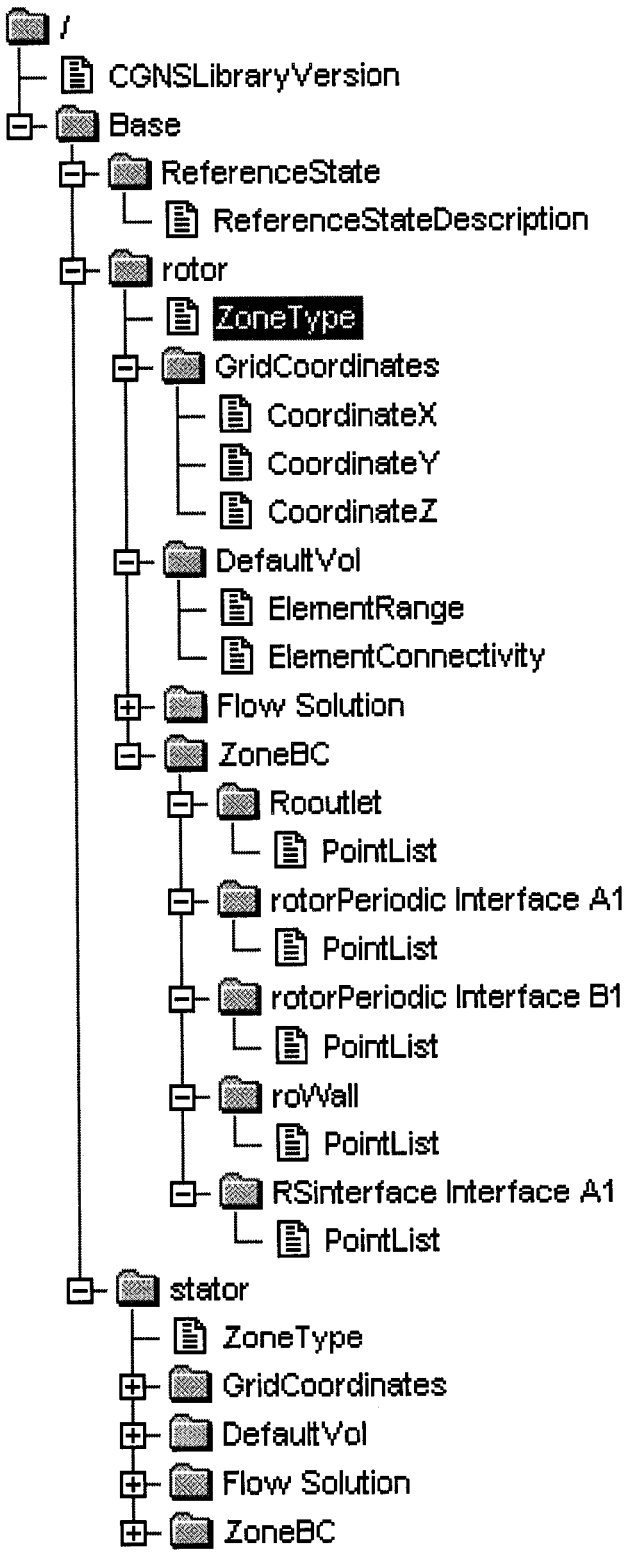
- On crée un domaine de type Interface nommé RSinterface qui est composée des frontières roInflow et stOutflow.

- Les paramètres utilisés pour définir cette interface sont :

- Le type d'interface = Fluid-Fluid avec l'option FrozenRotor

- L'axe de rotation Z

Des autres domaines qui seront définies comme étant de type interface avec le paramètre type d'interface périodique (Periodic), avec l'option Rotation et l'axe de rotation Z sont le rotorPeriodic et le statorPeriodic. Les deux derniers ont comme composantes les suivantes : roPeriodic1, roPeriodic2, stPeriodic1 et stPeriodic2.


/
CGNSLibraryVersion
Base
ReferenceState
ReferenceStateDescription
rotor
ZoneType
GridCoordinates
CoordinateX
CoordinateY
CoordinateZ
DefaultVol
ElementRange
ElementConnectivity
Flow Solution
ZoneBC
Rooutlet
PointList
rotorPeriodic Interface A1
PointList
rotorPeriodic Interface B1
PointList
roWall
PointList
RSinterface Interface A1
PointList
stator
ZoneType
GridCoordinates
DefaultVol
Flow Solution
ZoneBC

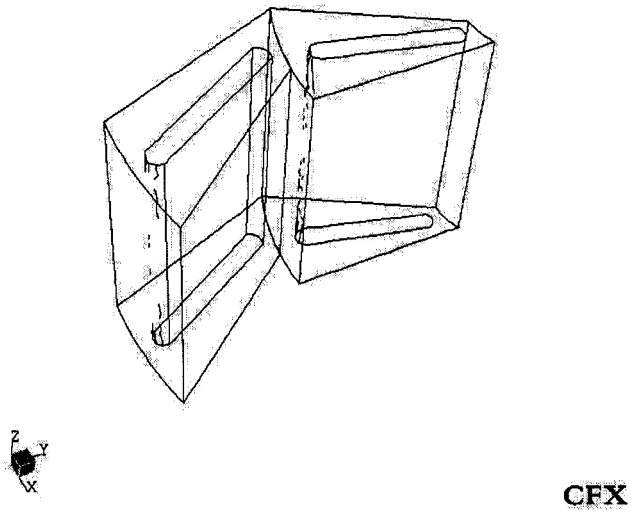


FIGURE III.2 La pale et la directrice

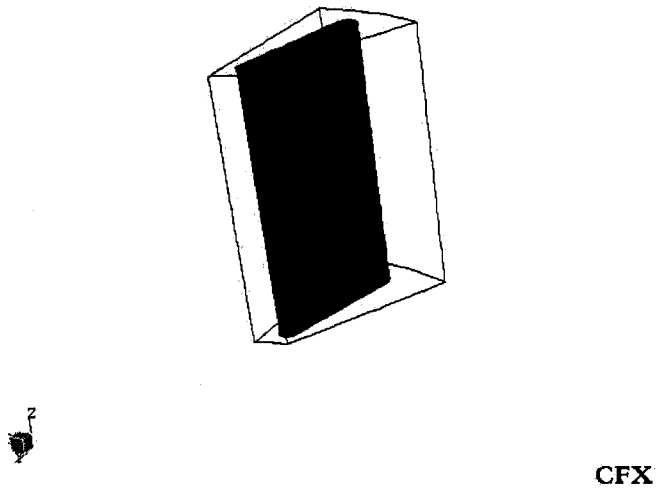


FIGURE III.3 La pale de la turbine dans le cas test (Rotor Stator)

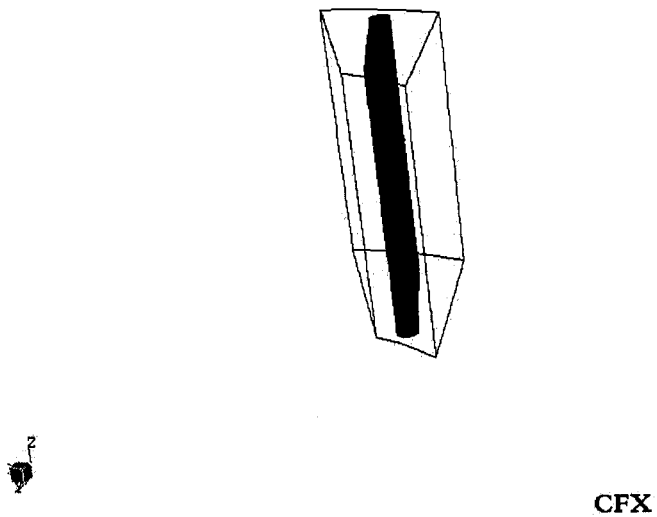


FIGURE III.4 Cas test CFX (Rotor Stator)

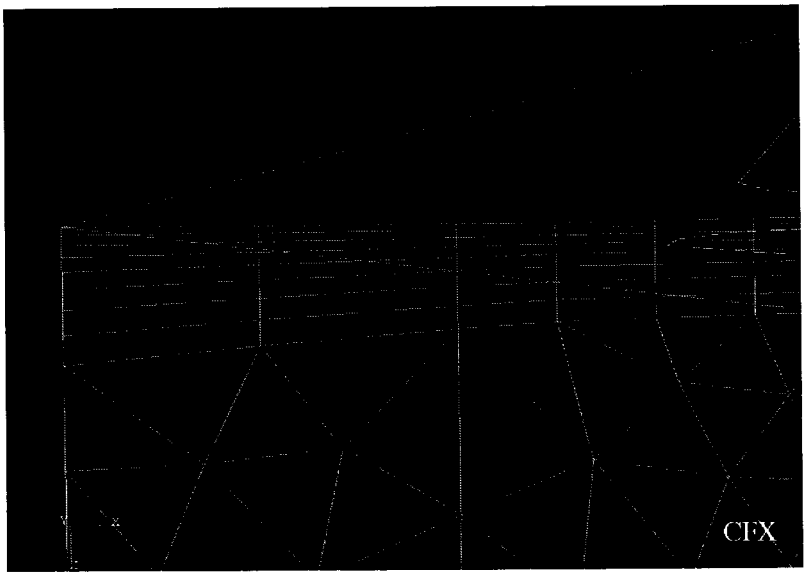


FIGURE III.5 La peau

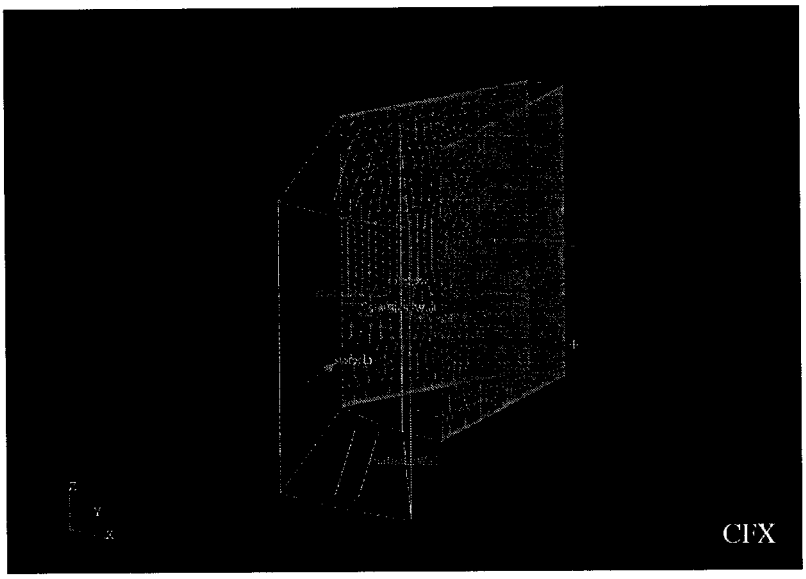


FIGURE III.6 La periodicite

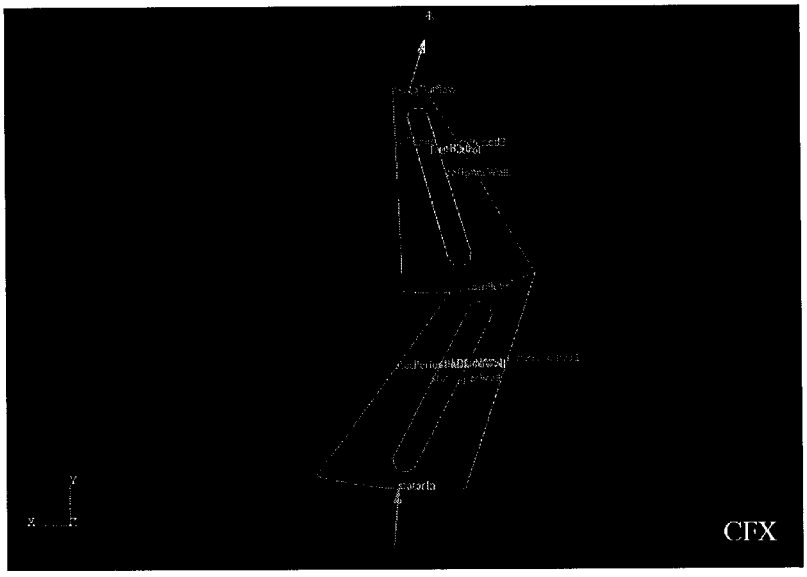


FIGURE III.7 L'interface entre fluides (section)

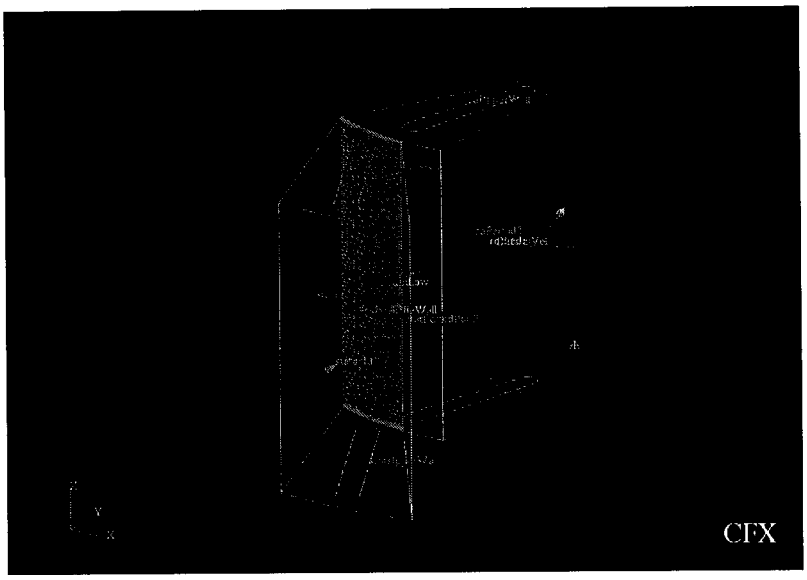


FIGURE III.8 L'interface entre fluides

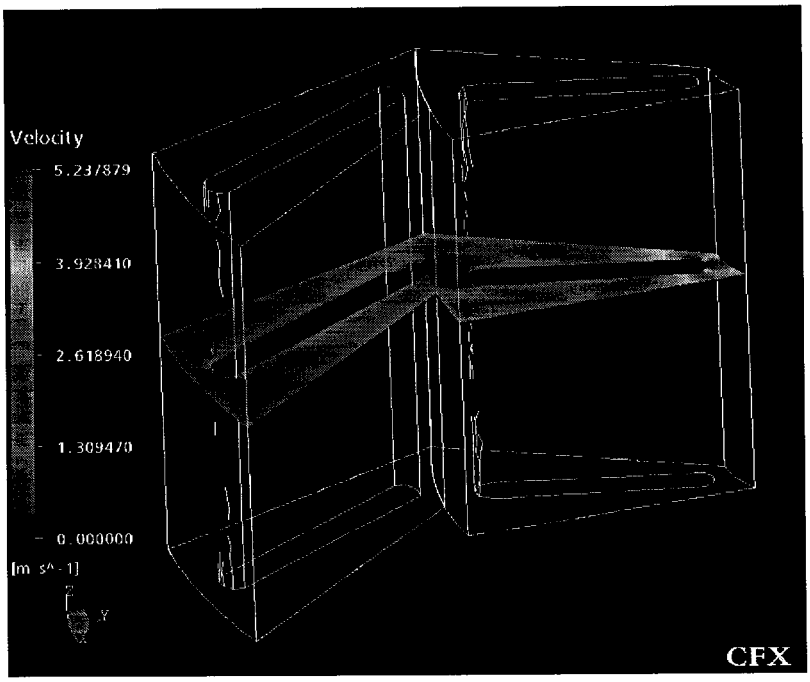


FIGURE III.9 La solution